MASTER IN COMPUTER SCIENCE

Visually effective Tropos models

Boucher, Quentin

Award date:
2008

Awarding institution:
University of Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Faculté d'informatique

Année académique 2007-2008

Visually effective *Tropos* models

Quentin Boucher

Mémoire présenté en vue de l'obtention du grade de maître en informatique.

Visually effective *Tropos* models

Quentin Boucher

Abstract

As part of the requirements engineering discipline, the purpose of goal modelling languages is to list the different goals of people fulfilled by the system to be developed. Those different goals and requirements are represented with diagrams in order to facilitate the developers-users communication. This is related to the fact that graphical representations are often considered as easier to understand than formal sentences. However, due to their complexity, those diagrams fail to reach their purpose.

In this thesis we analyse one goal modelling approach, called *Tropos* and its supporting tool *TAOM4e*. We apply the principles for effective communication which is measured as the speed, ease and accuracy with which the information content is understood. Following those principles the *Tropos* model can be decomposed into manageable modules; the most important entities can be emphasized so that they draw attention and are read first; visual variables can contribute to discriminate, identify and structure model elements in the diagram. The output of our analysis is a set of 3 recommendation lists. The first one is related to *Tropos* language engineers. The second is meant for developers of tools which, like *TAOM4e*, support *Tropos* diagrams creation. Finally, the last list includes recommendations for *Tropos* modellers on how to create and maintain a goal model using *Tropos* and *TAOM4e*.

Our suggested recommendations are validated in the illustrative example of the *Conference Management System*. In addition we have asked *TAOM4e* developers to evaluate the importance of our suggested recommendations. We hope that our proposal will contribute to improve *Tropos* and *TAOM4e* to create effective goal models.

Résumé

Les langages de modélisation des objectifs constituent une branche de l'ingénierie des exigences. De ce fait, ils visent à capturer les différents objectifs devant être satisfaits par le système à développer. Ceux-ci sont représentés sous forme de diagrammes afin de faciliter la communication développeurs-utilisateurs, but difficilement atteint en raison de la complexité de ces représentations graphiques.

Dans ce mémoire, nous analysons un langage particulier de modélisation des objectifs, appelé *Tropos*, associé à un logiciel, *TAOM4e* qui permet de construire des modèles dans ce langage. Nous appliquons des principes permettant d'atteindre une communication efficace, mesurée par la vitesse, la facilité et la précision avec laquelle le contenu informationnel est compris. Selon ces principes, un modèle *Tropos* peut être décomposé en de plus petits modules; les éléments les plus importants peuvent être mis en avant afin d'être lus en priorité; les variables visuelles peuvent contribuer à discriminer, identifier et structurer les éléments se trouvant dans un diagramme. De notre analyse résulte un ensemble de trois listes de recommandations. La première concerne les ingénieurs du langage *Tropos*. La deuxième est destinée aux développeurs d'outils, comme *TAOM4e*, qui supportent la création de diagrammes *Tropos*. Finalement, la dernière liste contient un ensemble de recommandations pour les modélisateurs *Tropos*, leur permettant de créer et mettre à jour un modèle d'objectifs en utilisant *Tropos* et *TAOM4e*.

Les recommandations proposées sont validées sur base d'un exemple de système de gestion de conférences. A notre demande, l'importance des recommendations pour les développeurs d'outils a été évaluée par les développeurs de *TAOM4e*. Nous espérons que nos propositions contribueront à améliorer *Tropos* et *TAOM4e* afin de créer des modèles d'objectifs efficaces.

Acknowledgements

I would like to acknowledge all the people supporting this master thesis. In particular, I wish to thank the following persons.

Prof. Anna Perini, for her hospitality during my stay in the *Automated Reasoning System* division of the *Fondazione Bruno Kessler* in Trento, for discussions about the topic of this thesis and also for her wise advises about the research process.

Mirko Morandini, Cu Nguyen Duy, Alberto Siena and Angelo Susi of the *Fondazione Bruno Kessler*, for their support in using the different tools, for their feedback about the different presentations made during my stay in Italy and for their active participation in the evaluation of the different recommendations proposed in this document.

Prof. Patrick Heymans and Dr. Raimundas Matulevičius, my supervisors at the University of Namur, for bringing the subject of this thesis to my attention, for helping me to fix the objectives of my research, for providing me the required documentation, for guiding me in the writing of this thesis and correcting its different versions.

Mr. Marc De Caluwe, for reading this document in order to correct its grammar and spelling.

Furthermore I would like to acknowledge the following people for their help in Italian everyday life: Chiara Di Francescomarino, Sapna Ponaraseri, Matteo Ceriotti, Leonardo Leiria Fernandes, Nauman Qureshi and Komminist Sisai.

I also have to thank the teachers, assistants and professors I met along my complete training, for sharing their knowledge with me and for building a part of what I am today.

Special thanks go to my parents and my sister for supporting and helping me, especially in more difficult periods. Thank you for always believing in me. Thank you also to my friends for giving me the required energy and for always being there for me, also in very difficult moments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The success of software systems depends on the degree of satisfaction of their users. Indeed, those artifacts are generally made to fulfil a certain need of a given user. And if the developed solution doesn't help her/him, s/he won't use it. But, generally, there are misunderstanding problems between software designers and software users. It implies that developers develop software which don't match with users' requirements [29]. As the developed application doesn't help them, users won't use it.

The goal of the *software systems requirements engineering* discipline is to reduce this phenomenon of "useless" software [23]. Its purpose is to have a complete understanding of stakeholders' needs in order to take them into account. The activities included in the requirements engineering process try thus to identify the different stakeholders, their needs, their resources, *etc.* This information has to be collected from different stakeholders who are, from near or from far, concerned by the developed system. However, one generally has some difficulties to get information from the different stakeholders because they generally really don't know how to express their needs and requirements on the needed system. Moreover, they aren't used to those methods of requirements engineering. They are generally specialised in their domain and know how to carry out their job but have difficulties to express it in a formal way.

In order to get this information easier, one can, for example, use requirements engineering techniques such as use cases [25], problem frames [13], feature diagrams [14], *etc.* One of those is the *goal modelling* technique [1]. The objective of goal modelling languages is to understand the different goals of stakeholders and express them in a graphical way. Moreover, like the general techniques, they also aim at facilitating the communication between system developers and stakeholders of the modelled domain. Goal modelling languages are generally graphical ones. This way one is able to discuss with a graphical representation about the stakeholders' goals. One generally considers that it is easier to communicate with a graphical representation than with natural language sentences containing the same information. This is probably related to the belief that graphics are more direct and don't add useless information [26].

Over time, several goal modelling languages have been developed. We can, for example, mention *i** [32], *KAOS* [31], *NFR* [8] and *Tropos* [6]. Obviously, each of those languages comes with its own syntax, semantic and terminology. However, although focusing on different development stages [15], their goal is the same: capture users' requirements in order to develop an information system adapted to the needs.

In our thesis, the focus is on the relationship between the visual aspects of goal modelling diagrams and their understanding by humans. Those lasts are generally the stakeholders of the domain of the system to develop. Consequently, it is important that they understand the information presented in a certain schema which concerns them directly or not. The fact that communication is easier with graphics is largely based on intuitions and slogans like "a picture is worth a thousand words" [21]. Their quality depends on a large set of factors like software engineers' skills. However, Moody claims that diagrams act "as a barrier rather than an aid to user-developer communication" [21]. Consequently, in [21], a set of 9 principles which aims at making diagrams communicating effectively is presented. They come from a wide range of disciplines like cartography, cognitive integration, communication theory, *etc.* It implies that it takes a lot of aspects of human mind into account.

This thesis focuses on diagrams of a specific goal modelling language, namely *Tropos*. We analyse how *Tropos* fulfils the different principles presented in [21]. This analysis is done with a tool which allows one to draw diagrams of this language: *TAOM4e* (Version 0.5.0). Indeed, one requires such a tool in order to build *Tropos* diagrams. With this analysis, we can see if *Tropos*/*TAOM4e* allow one to create and maintain effectively communicative models. The objective of this analysis is to formulate lists of recommendations for language engineers, tool developers and modellers. These lists aim at modifying *Tropos*, *TAOM4e* and modellers habits in order to build effective diagrams according to the definition of such diagrams provided in [21]. Moreover, these lists are also validated in two ways. The first technique is based on the illustration of the application of all recommendations together on a single example. The second technique relies on *TAOM4e* developers' opinion about recommendations for tool developers.

1.1 Structure

This work is divided in three parts as follows.

Part I introduces the background of our work. It consists of Chapter 2 which presents the *Tropos* goal modelling language, Chapter 3 which presents the tool which we evaluate: *TAOM4e*. Finally, Chapter 4 presents the 9 principles for effective communication [21].

Part II presents our contribution based on elements presented in Part I. In Chapter 5, we analyse *Tropos*/*TAOM4e* with respect to the 9 principles provided in [21]. The list of recommendations is presented in the 3 next chapters. Chapter 6 is related to recommendations for *language engineers*, Chapter 7 includes recommendations for *tool developers* while Chapter 8 finally presents

recommendations for *modellers*.

Part III presents the validation of the recommendations explained in the previous part. In Chapter 9, the validation of the three types of recommendations is made on basis of a *Conference Management System* example. While in Chapter 10, the validation of recommendations for tool developers is made by collecting opinions from *TAOM4e* developers.

PART IV finally presents the conclusions of our work. Moreover, we also present its limitations and potential future work.

The complete structure and the flows between the different chapters are summarized in Figure 1.1.

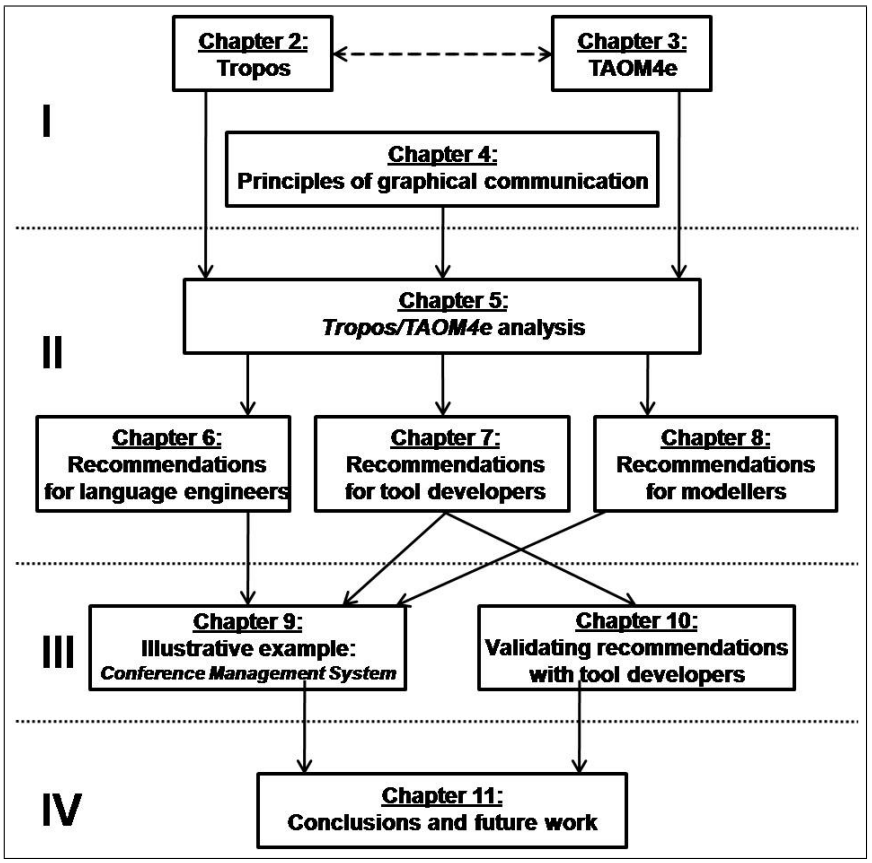


Figure 1.1: Thesis structure

1.2 Running example

In order to illustrate the different aspects of *Tropos*, *TAOM4e* and principles of graphical communication [21] presented from Chapter 2 to Chapter 8, we use a running example which is called *Meeting Scheduling System* [10, 16, 32].

The goal of this system is to schedule meetings and support their setting up. The requirement for the computer-based meeting scheduler is that, for each meeting request, it should try to determine its date and location. Moreover those values should be chosen in such a way that they maximize the number of intended participants.

The two actors of the current domain are the Meeting initiator and the Meeting participant. Without a computer-based meeting scheduling system, the Meeting initiator has to ask the availability of each potential participant. The dates depend on the personal agendas. The Meeting participant provides thus a range of preferred dates but s/he also gives the set of dates on which s/he can't attend the meeting. Once the Meeting initiator has received all preferred and exclusion dates, s/he proposes a date which isn't included in any exclusion set and which belongs to as many as possible preference sets. The potential participants have then to give their agreement about the proposed date.

This example will be used to illustrate the different elements presented in Parts I & II. The diagrams will include the domain as it is currently (*as-is*) but also including the Meeting scheduler (*system-to-be*). The role of this Meeting scheduler is to manage the exclusion and preference dates in order to automatically determine the optimal date.

Part I

Background

Chapter 2

Tropos

Tropos is a software development methodology which aims at building agent oriented systems [6]. It is based on *i** [32] which offers the notions of actor, goal and (actor) dependency. Those notions are used to model early and late requirements, architectural and detailed designs.

In this chapter we are thus going to present the concepts of *Tropos* which will be used in the other chapters of this study. In order to do this, we first present, in Section 2.1, the different *Tropos* phases. Then, in Section 2.2, the different modelling activities and *Tropos* diagrams are presented. Finally, in Section 2.3, we present the meta-model of *Tropos*.

2.1 *Tropos* phases

Tropos is composed of different phases [6]. There are actually 5 phases: *Early requirements*, *Late requirements*, *Architectural design*, *Detailed design* and *Implementation*. It means that the *Tropos* methodology is supposed to "support all analysis and design activities in the software development process, from application domain analysis down to the system implementation" [6]. The idea underlying *Tropos* is to build a model of the environnement and the *system-to-be* which is incrementally refined through the different phases. Hereunder, we are thus presenting those different *Tropos* phases.

The first *Tropos* phase is thus the *Early requirements* one. This first phase is "well accepted in the Requirements Engineering research community, but not widely practised" [6]. It is part of the requirements analysis which is generally the initial step in many software engineering methodologies. During this phase, the different *stakeholders* are identified. *Stakeholders* are generally actors, or more generally entities, which are involved in the domain of the software system. Those *stakeholders* are modelled as social actors who depend on each others. An actor can depend on another for goals to be achieved, plans to be performed and resources to be supplied. Those dependencies help to know *why* system functionalities have to be developed. Indeed, with those relationships, one understands *why* an actor depends on another. Moreover, at the end of the development, those statements can help to know if the final implementation

meets the initial requirements. This phase of the *Tropos* methodology aims thus at defining the system *as-is*. Indeed, in the *Early requirements* phase, only the stakeholders involved in the current system are represented. The output of this phase is an organisational model which includes the relevant actors and the interactions among them.

In the second *Tropos* phase, the *Late requirements* one, the model built during the previous phase is extended with a new actor: the *system-to-be*. Moreover, the dependencies between this new actor and the ones modelled in the *Early requirements* phase are added. These dependencies define the functional and non-functional requirements of the *system-to-be*. The *Early requirements* and *Late requirements* phases compose the two parts of the requirements analysis.

Next phases are *Architectural design* and *Detailed design* where the emphasis is placed on the *system-to-be*.

The *Architectural design* phase defines the architecture of the *system-to-be* in terms of subsystems. Those subsystems are interconnected through data and control flows. Subsystems are represented as actors while the data and control flows are represented as dependencies. During this phase, an assignation of the different agents and their capabilities are also defined.

In the *Detailed design* phase, the internal structure of the different agents identified in the previous *Tropos* step is defined and the different interactions of those agents are specified. At this point, the development platform has already been chosen. It implies that the agents and their interactions are defined according to the chosen implementation language.

The final step is the *Implementation* phase. This implementation is made in the development language chosen during the previous phase. One has thus to follow the design specifications defined in the *Detailed design* phase.

2.2 Modelling activities

We can still be more precise by explaining the five different modelling activities which help in building the final model: *actor modelling*, *dependency modelling*, *goal modelling*, *plan modelling* and *capability modelling*. In this section we explain thus those five different modelling activities.

During the *actor modelling* step, the different stakeholders are identified and analysed [6]. Those stakeholders can be actors of the environment or system actors or agents. As it was explained in the previous section dedicated to *Tropos* phases, if one is modelling an *Early requirements* diagram, one focuses on stakeholders of the domain, their intentions, *etc.* While if one is building a *Late requirements* diagram, one focuses on the *system-to-be* actor. And so on.

The *dependency modelling* consists of "identifying actors which depend on one another for goals to be achieved, plans to be performed and resources to be furnished" [6]. So, for example, during the *Early requirements* phase, one has to focus on the dependencies among the different actors of the system *as-is*. Dependencies can also be added later, depending on the goals identified dur-

ing the *dependency modelling* activity explained in the next paragraph. During the *Late requirements* phase, the *system-to-be* is added. Consequently, some relationships between this new actor and the ones identified during the *Early requirements* phase have to be added. They express the goals, plans and tasks that domain actors can expect from the *system-to-be*, and conversely. During the *Architectural design* phase, one models the data and control flows between the different subsystems of the *system-to-be*. Obviously, those flows are a kind of dependency.

A diagram resulting from the *actor modelling* and *dependency modelling* is called an *actor diagram*. Such a diagram is defined as "a graph where each node represents an actor and each arc represents a dependency between the two connecting nodes" [6]. An arc has a label which represents the object of the dependency. This object can be a *goal*, a *task* or a *resource*.

Figure 2.1 is an example of *actor model* of our *Meeting Scheduling System* example. There are three actors (represented as circles): Meeting Initiator, Meeting Participant and Important Participant. The links between them represent *goal* and *resource* dependencies. So for example, the Meeting Initiator depends on the Meeting Participant through the Attends Meeting goal (represented as an oval). Conversely, the Meeting Participant depends on the Meeting Initiator to get the Proposed Date(m) resource (represented as a rectangle). All those constructs will be discussed further in this chapter.

Another activity is the *goal modelling*. During this step, the focus is on actors' goals analysis. Three reasoning techniques are used: *means-end analysis*, *contribution analysis* and *and/or-decomposition*. *Means-end analysis* aims at finding the different *plans*, *resources* and *goals* which can achieve a *goal*. The *contribution analysis* aims at finding the *goals* which can contribute, positively or negatively, to a given actor's goal. The *and/or-decomposition* aims at decomposing the original goal into sub-goals.

The *goal modelling* is applied to three *Tropos* phases: *Early requirements*, *Late requirements* and *Architectural design*. During the two first phases, the *goal modelling* is used to refine the goals of the different domain actors and the *system-to-be*. During the *Architectural design* phase, this modelling technique is used in order to decompose the *system-to-be* in a set of sub-actors.

Another analysis technique is the *plan modelling*. As it is clear from its name, this analysis focuses on *plans*. This technique is the same as the *goal modelling* excepted that the object of the analysis is the *plan*. It implies that the reasoning techniques are the same: *means-end analysis*, *contribution analysis* and *and/or-decomposition*.

A *goal diagram* is a graphical representation of *goal* and *plan modelling* [6]. Actually it represents the point of view of an actor. It is represented as a circle, representing the *"holds" relationship*, containing *goals*, *plans* and *resources* of the actor. Those constructs are linked by arcs which represent *means-end*, *contribution* and *and/or-decomposition* relationships.

An example of *goal diagram* for the Meeting initiator actor of our *Meeting Scheduling System* example is depicted in Figure 2.2. This actor has one major goal, Organize meeting, which is decomposed, with an *and-decomposition*, in two

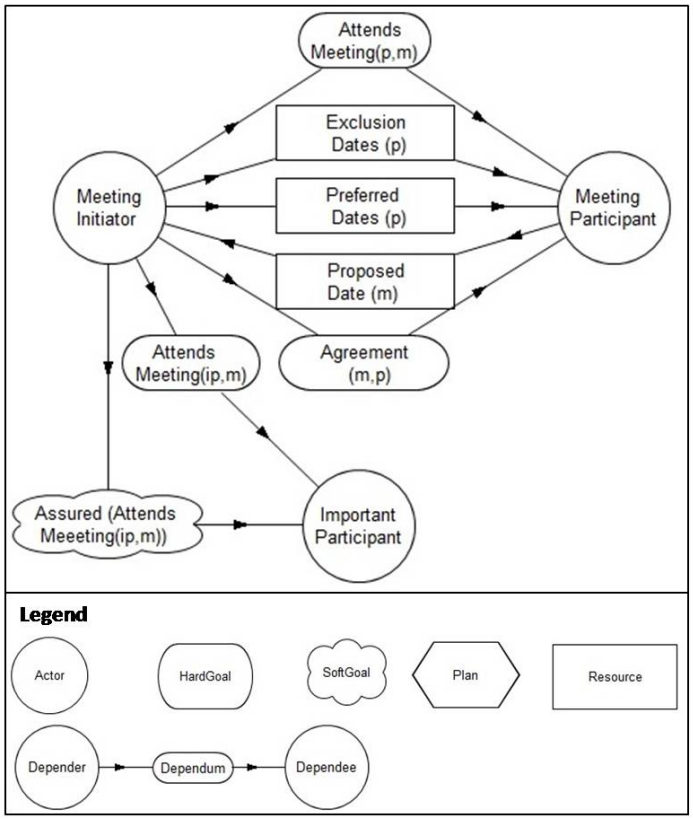


Figure 2.1: An example of *actor diagram* (adapted from [32])

sub-goals: Invite attendees and Meeting BeScheduled. As it is indicated with the *means-end link*, the Meeting BeScheduled hardgoal is achieved by the Schedule meeting plan. This plan is, in turn, decomposed in three sub-plans with an *and-decomposition*. Its sub-elements are Find agreeable slot, Get preferred dates and Get exclusion dates plans. One can also notice that the Find agreeable slot plan contributes positively to the Maximize the number of attendees softgoal (represented as a cloud).

The last activity is the *capability modelling*. It starts at the end of the *Architectural design* phase, just after having defined the goals and dependencies of the different sub-systems. During the *capability modelling*, the capabilities of the sub-systems are delimited. Those capabilities represent "the ability of an actor of defining, choosing and executing a plan for the fulfilment of a goal, given certain world conditions and in presence of a specific event" [6]. It implies that *goals* and *plans* modelled in the previous phases become part of the capabilities.

A graphical representation of the capabilities is given by *capability* and *plan* diagrams. But we won't explain them here. We just mention that *UML activity diagrams* and *AUML action diagrams* are generally used to represent those types of diagrams [6].

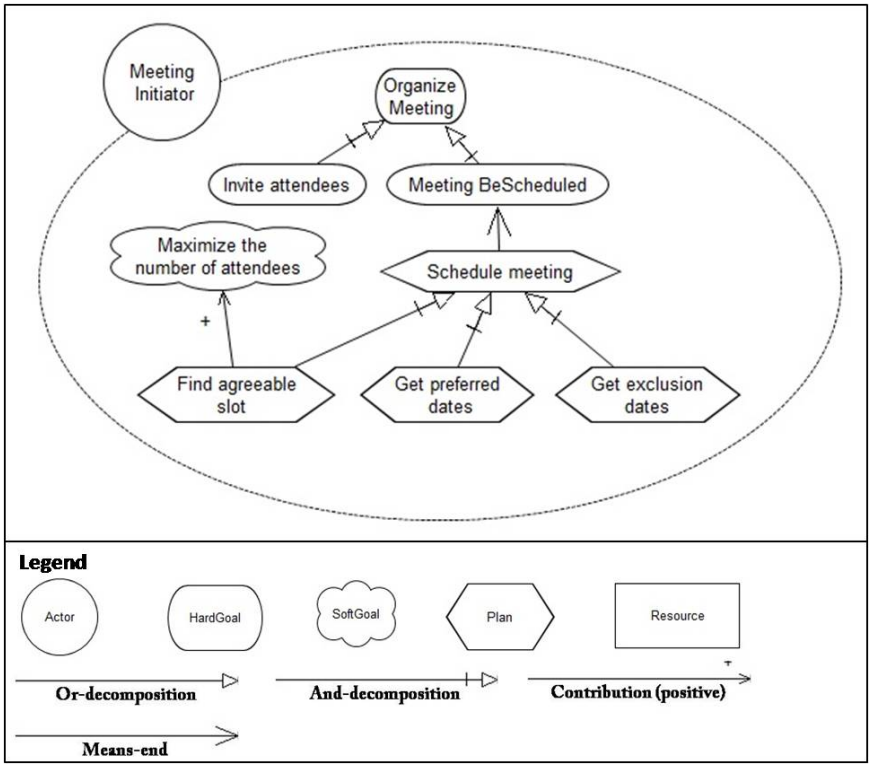


Figure 2.2: An example of *goal diagram* (adapted from [32])

2.3 The meta-model

The meta-model of a modelling language, including *Tropos*, can be defined by an architecture composed of 4 different layers [6]: the *meta-meta-model*, the *meta-model*, the *domain* and the *instance*. They are summarized in Table 2.1.

In this section, the *meta-model* layer is presented and illustrated with the *Meeting Scheduling System* example, in other words with a *domain* layer. The presentation structure is quite the same as in [6], [11] and [27]: Section 2.3.1 presents concepts related to *actor diagrams* while in Section 2.3.2, concepts linked to *goal diagrams* are explained.

In all those sections, the meta-models are presented in terms of UML class diagrams.

2.3.1 Concepts related to actor diagrams

An actor is an entity which has strategic objectives motivating it to participate into the domain or the new system. Figure 2.3 represents the portion of the *Tropos* metamodel related to actor diagrams presented in Section 2.2.

There, one can see that Position, Agent and Role are specializations of the concept of Actor. Those concepts have to be defined. Actually, an actor "rep-

Table 2.1: The 4 layers architecture of the *Tropos* meta-model

Level	Description	Examples
Meta-meta-model	Specifies language structural elements	Attribute, Entity
Meta-model	An instance of the meta-meta-model. It's the core of the language	Actor, Goal, Plan
Domain	An instance of the meta-model. Define a model of the domain	Meeting Initiator, Meeting Participant
Instance	An instance of a domain element	J. Smith: instance of Meeting Participant

resents a physical, social or software *agent* as well as a *role* or *position*" [6]. A *software agent* is characterized by properties like autonomy, reactivity, proactivity, social capabilities as it is explained in [24]. While a *role* is an abstract definition of the compportment of a social actor inside the application domain. Finally, a *position* represents a set of *roles* generally played by one *agent*. It means that an *agent* can occupy a *position* which can cover several *roles*. This phenomenon is expressed in the UML class diagram of Figure 2.3. Indeed, one can see that an agent can occupy 0..n *position(s)* and play 0..n *role(s)*. While a *position* covers 1..n *role(s)*.

One can also read that an *actor* can have 0..n *goal(s)* which can be *Hardgoal(s)* or *Softgoal(s)*. Conversely a *goal* is wanted by at least one actor. The difference between the two types of goals is that *Hardgoals* have precise criteria determining if they are satisfied while *Softgoals* don't have such criteria. One generally says that *Hardgoals* are *Satisfied* while *softgoals* are *satisficed*. *Softgoals* are generally used to model, for example, the security, usability and performance aspects.

An actor *dependency* is a quaternary relationship including two instances of the Actor class, one of the Goal, Plan or Resource class and, optionally, a reason for the dependency (labelled *why*). A dependency between two actors indicates that one actor depends, for some reason, on the other one in order to reach a *goal*, satisfy a *plan* or get a *resource*. The first actor is called the *depender*, the second, the *dependee* and the object of the dependency, the *dependum*.

An example of actor diagram related to our *Meeting Scheduling System* example is depicted in Figure 2.4. It is the actor diagram of the *Late requirements* phase. All the constructs presented in this diagram are instances of the meta-model of Figure 2.3. Three actors, represented as circles, are illustrated: the Meeting Initiator, the Meeting Participant and the Meeting Scheduler. There are some dependencies between them. So, for example, the Meeting Initiator depends on the Meeting Participant. The first one expects that the second will

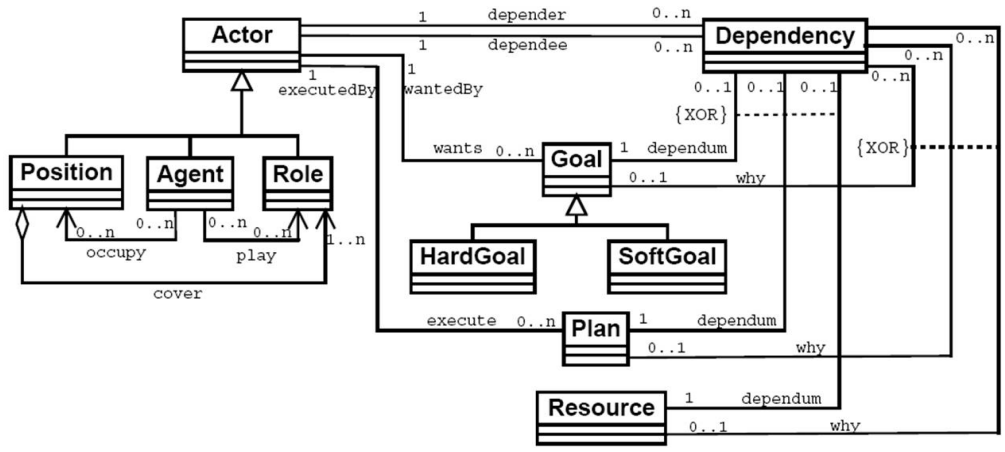


Figure 2.3: UML class diagram of the concepts related to *actor diagrams* (from [11])

Attend meeting (*Hardgoal* (represented as an oval) dependency). In this case, the Meeting Initiator is the *depender*, the Meeting Participant, the *dependee* and the Attend meeting *Hardgoal* the *dependum*. There is also a *resource* dependency, Proposed Date (represented as a rectangle), between Meeting Participant and Meeting Scheduler. In this case, the Meeting Scheduler is the *dependee*, the Meeting Participant, the *depender* and the *dependum* is the Proposed Date resource.

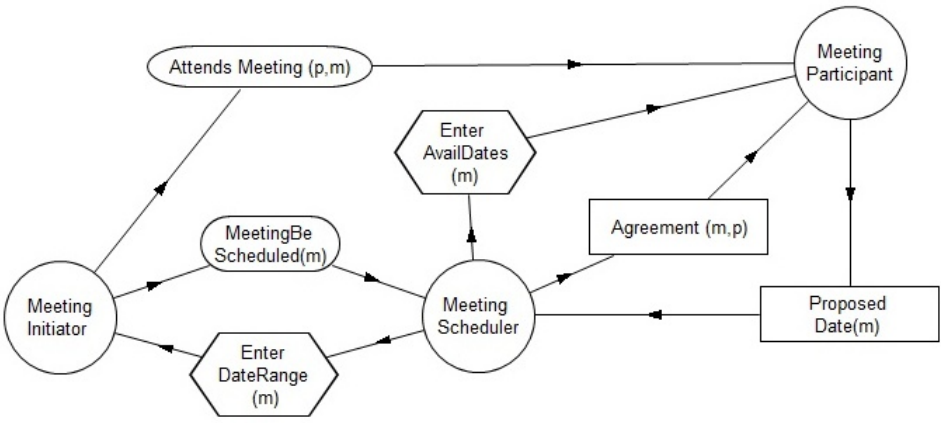


Figure 2.4: The *Tropos Late requirements* diagram of the *Meeting Scheduling System* example

2.3.2 Concepts related to goal diagrams

The second type of *Tropos* diagram is the *goal diagram*. As *goal diagrams* are the result of *goal modelling* and *plan modelling* activities, *goal* and *plan* analysis are the central elements of both meta-models presented in this section. Figure 2.5 represents the meta-model of the concepts related to *goal analysis* while Figure 2.6 represents the meta-model related to *plan analysis*.

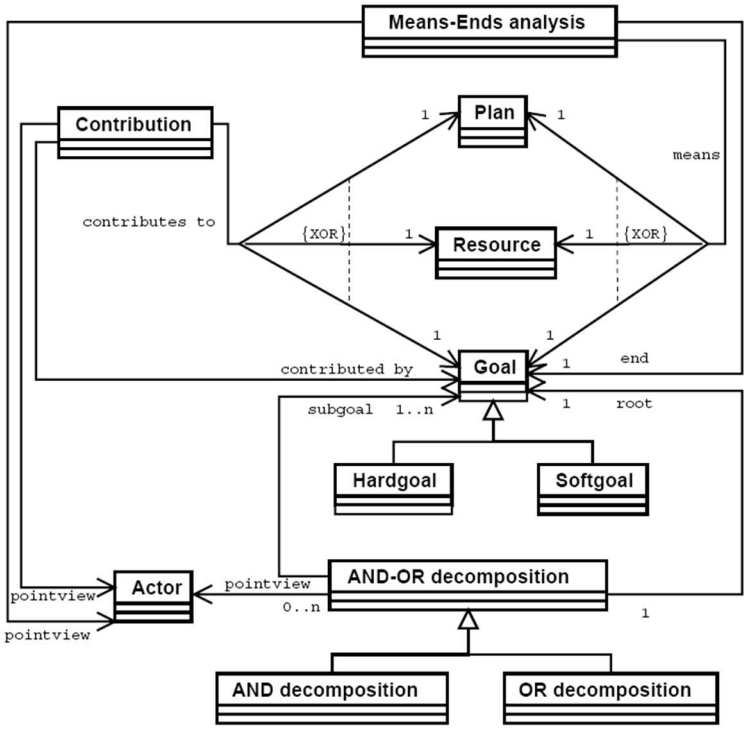


Figure 2.5: UML class diagram of the concepts related to *goal analysis* (from [6])

We first focus on the *goal* aspect. In the meta-model of Figure 2.5, this *goal* is represented by the Goal class which is, as in the meta-model of Figure 2.3, specialized in Hardgoal and Softgoal sub-classes. As it was explained in Section 2.2, there are reasoning techniques: the *means-end analysis*, the *contribution analysis* and the *and/or-decomposition*. As one can notice, those three techniques are represented in the meta-model of Figure 2.5. They all depend on the point of view of the *actor*. This is represented by the association, labelled pointOfView, between the Actor class and Contribution, Decomposition and Means-End Analysis classes.

The *means-end analysis* is a ternary relationship. It includes the *actor* whose point of view is represented, a *means* and an *end*. The *end* is mandatorily a *goal* while the *means* can be, as it is visible in the UML class diagram, a *plan*,

goal or *resource*.

The *contribution* is also a ternary relationship. It still includes an *actor*, whose point of view is represented, a *goal* and a *goal*, *plan* or *resource*. The first *goal* is the contributor. It is represented by the contributed by name of the association between Goal and Contribution classes. The element at the other extremity of the contribution link can be a *goal*, *plan* or *resource*. It is represented by the association labelled contributes to between contribution and goal, plan and resource classes. Contribution links indicate that a *goal* contributes positively or negatively to the fulfilment of another *goal*, *plan* or *resource*. In order to see the influence of a *goal*, values can be associated to contribution links. Those values are *+*, *++*, -, --. The *++* value associated between *g1* and *g2* means that, if *g1* is satisfied, then *g2* is satisfied. A *+* value means that the source *goal* contributes partially to the fulfilment of the target goal, plan or resource. -- and - values have the dual effects.

Finally, the *decomposition* link is also a ternary relationship still including the *actor* whose point of view is represented. The two other constructs are *goals*. One is labelled as the root and the other as the subgoal. There are two types of decompositions: *And* and *Or*. If a *goal* is decomposed in several sub-goals with an *and-decomposition*, all those sub-goals must be satisfied in order to satisfy the root *goal*. While in an *or-decomposition*, if a single sub-goal is verified then the root *goal* is satisfied. A *goal* decomposed in a single sub-goal is equivalent to a *++* contribution.

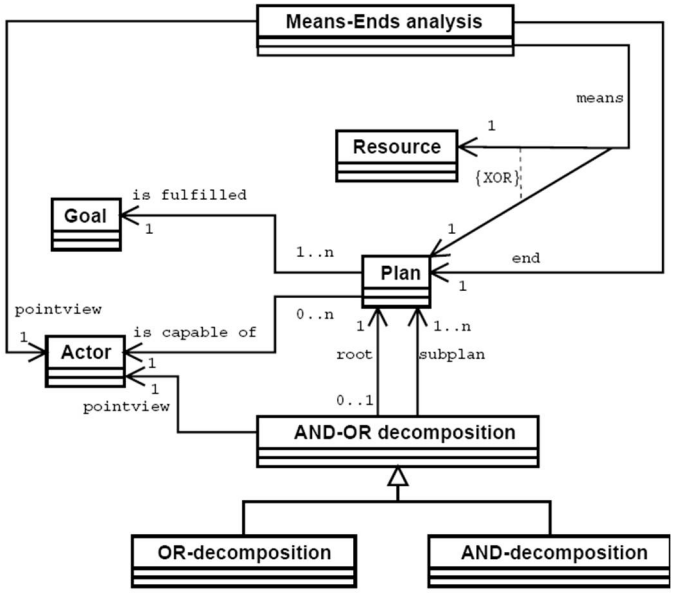


Figure 2.6: UML class diagram of the concepts related to *plan analysis* (from [6])

The situation of the meta-model focusing on the *plan* construct of Figure 2.6 is quite the same as the one of Figure 2.5. Two reasoning techniques are still present: *means-end analysis* and *and/or-decomposition*.

2.4 Summary

In this chapter we presented *Tropos* which is a software development methodology which uses the notion of *agent* in all development phases. We explained thus the different *Tropos* phases and their respective roles. Then, the different modelling activities and the resulting *Tropos* diagrams were presented. Finally, we also explained the *Tropos* meta-model.

Chapter 3

TAOM4e

In this chapter, we analyse *TAOM4e*, a tool supporting *Tropos*. Our analysis is structured as follows. First, in Section 3.1, we discuss general aspects of tools for modelling *i* / Tropos* diagrams. In Section 3.2, we give an overview of the environment of *TAOM4e*. Then, in Section 3.3, we describe the general architecture of the tool. Finally, in Section 3.4, we show how to use the tool. In order to do this, we present the installation of *TAOM4e*, the creation of a project and the different windows and tabs of the tool.

3.1 Modelling tools for *i* / Tropos*

One is able to draw *i* / Tropos* diagrams with tools like, for example, Dia [9]. However this solution has some drawbacks. First, using this type of tool can be time-consuming for the modeller. Indeed, s/he is not helped in the drawing activity and has to draw the different constructs herself/himself. It means that s/he always has to re-draw the same elements. It is time-consuming and can be very boring to always make the same things. Moreover, the fact that modellers have to draw the different *i* / Tropos* constructs themselves implies that they are free to do whatever they want. It means that there is no mechanism checking if the models built by modellers respect the meta-model. Consequently diagrams can have illegal constructs according to this meta-model. This situation can also be disturbing for modellers because they always have to pay attention to what they make and refer to the meta-model.

The solution to those problems could be a tool supporting the drawing of *i* / Tropos* diagrams. This kind of artifact facilitates modellers' work. Indeed, it provides them an automated way of building the different constructs which can be inserted in an *i* / Tropos* diagram. Moreover, this type of tool is generally built according to the meta-model of the language. This property implies that modellers aren't able to do whatever they want. They are limited by the tool which implements the meta-model. Using this kind of tool prevents modellers from building incorrect models and from wasting time in drawing constructs.

TAOM4e [4], *Si*-Tool* [5], *OME* [30] are examples of such tools for *i* / Tropos*. In the next chapters, our analysis is carried out with *Tropos*. So, in order to get readers' understanding, we present, in this chapter, its most important elements. However, we don't present everything about this tool. We focus on aspects which are used further in the text. The goal is that readers understand basic elements of *TAOM4e* which are used in the different chapters, no more.

3.2 General presentation of *TAOM4e*

Tool for Agent-Oriented Modelling for *Eclipse* (*TAOM4e*) is a tool supporting the *Tropos* methodology. The tool is made for the *eclipse* platform[1]. It has been developed as an extension, also called *plug-in*, of this platform. An *eclipse* plug-in is the smallest unit or function. Figure 3.1 depicts the *eclipse* plug-in architecture. One can see that a plug-in is plugged on top of the Platform run-time, the Workbench and the Workspace of *eclipse*. The Workbench and the Workspace are components of *eclipse* which won't be explained here. *Eclipse* is made of a set of plug-ins.

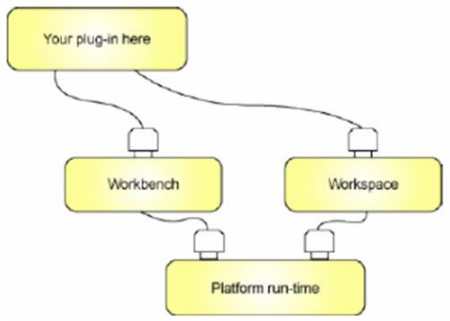


Figure 3.1: *Eclipse* plugin architecture (from [4])

TAOM4e is a project of the Software Engineering research line at the *Fondazione Bruno Kessler* (former *Istituto Trentino di Cultura*) in Trento, Italy[2]. The current version available on the website of the tool is 0.5.0. Consequently, we used this version for our work.

3.3 General architecture of *TAOM4e*

Figure 3.2 depicts the component view of *TAOM4e*. Actually, this figure is an instance of Figure 3.1 with *TAOM4e* plug-in. Indeed, one can see that Platform run-time, Workbench and Workspace are still present. *TAOM4e* is composed of two elements: *TAOM4e* platform and *TAOM4e* model. The *TAOM4e* platform represents the plug-in itself while the *TAOM4e* model represents the

[1] See `http://www.eclipse.org`
[2] See `http://sra.itc.it/tools/taom4e/`

implementation of the *Tropos* meta-model presented in Chapter 2. Two other plug-ins, EMF and GEF, are situated between the two components of *TAOM4e* and the ones related to the *eclipse* platform.

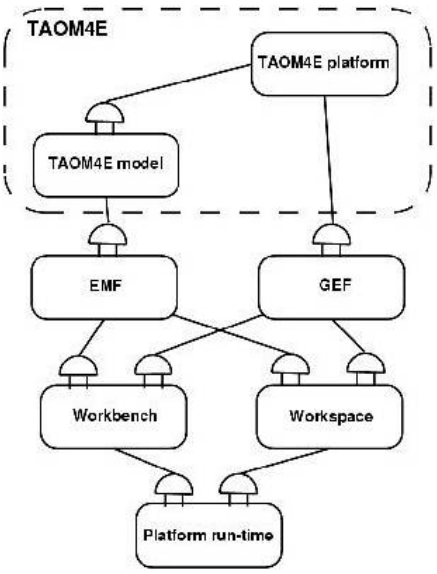


Figure 3.2: *TAOM4e* component view (from [4])

GEF stands for *Graphical Editing Framework*. It is an open source framework which helps in creating graphical editors within *eclipse*. The most important features of this framework are to display a model graphically, allow the user to interact with that model, process and interpret user input from mouse & keyboard and provide mechanisms for updating the model [4].

EMF stands for *Eclipse Modelling Framework*. It is a framework which helps to easily transform models into efficient, correct and easily customizable Java code [4].

3.4 Using *TAOM4e*

In this section we explain how one has to use *TAOM4e*. In order to do this, we first explain the steps required for the installation of the tool. Then, we give the different actions required in order to create a *Tropos* project in *TAOM4e*. And, finally, we explain the different elements of its graphical interface.

3.4.1 Installation

One has to install *TAOM4e* first. As it is an *eclipse* plug-in, the way to install it is the same as other plug-ins of this platform. However, a preliminary step is required. In Section 3.3, we saw that *TAOM4e* was built on top of *GEF* and *EMF*. Consequently one has to download the zip archives of each of those plug-ins. Then, one can install them by copying the content of *Features* and *Plugins* directories in the folders of the *eclipse* platform having the same

names. It means, that the content of *Features* and *Plugins* directories of zip archive will be copied into, for example, "C:/Program Files/Eclipse/features" and "C:/Program Files/Eclipse/plugins". Then, one has to copy the *Plugins* folder of *TAOM4e* into the same folder as for *EMF* and *GEF*.

3.4.2 Starting a new *Tropos* project in *TAOM4e*

When the different installation steps are finished, one can launch *eclipse*. One is now able to create *Tropos*/*TAOM4e* projects. In order to do this, one has to select menu File→New→Project, choose a General project and click the Next button. In the next window, one has to give a name to the project, for example "Meeting Scheduling System", and click the Finish button.

The project has been created and is visible in the *Navigator* tab of *eclipse* (see Figure 3.3). Now, one has to right-click on the created project, in our case "Meeting Scheduling System", navigate to New→Other, select TAOM4E platform→Tropos Model and click Next button. In the next window, one has to give a name to the *Tropos* project, for example "MSS", and then click Finish. The new project is created and visible in the *Navigator* tab. Moreover, the *TAOM4e* perspective, which is a way of presenting the different tabs, is automatically activated.

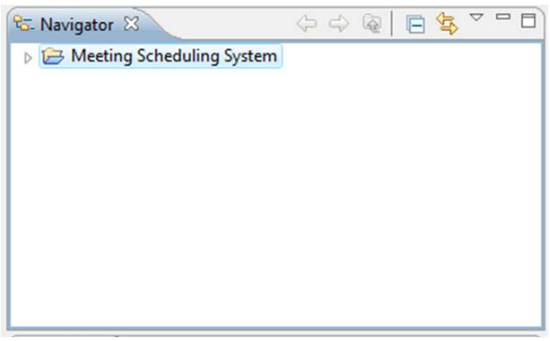


Figure 3.3: *Eclipse* navigator tab

3.4.3 Components of *TAOM4e*

The graphical user interface (GUI) resulting from the first steps presented in the previous section is depicted in Figure 3.4. This interface is composed of 4 tabs: the *Navigator* (Part A), the *Outline* (Part B), the *Properties* (Part C) and *MSS.tropos* (Part D).

We already spoke about the *Navigator* tab (Part A of Figure 3.4) in the previous section (see Figure 3.3). Actually, this part isn't specific to *TAOM4e* but to *eclipse*. It presents the different projects and the different files associated to each of them.

The *Outline* tab (Part B of Figure 3.4) shows the different *Tropos* phases. Indeed, Early Requirements, Late Requirements, Architectural Design, Detailed

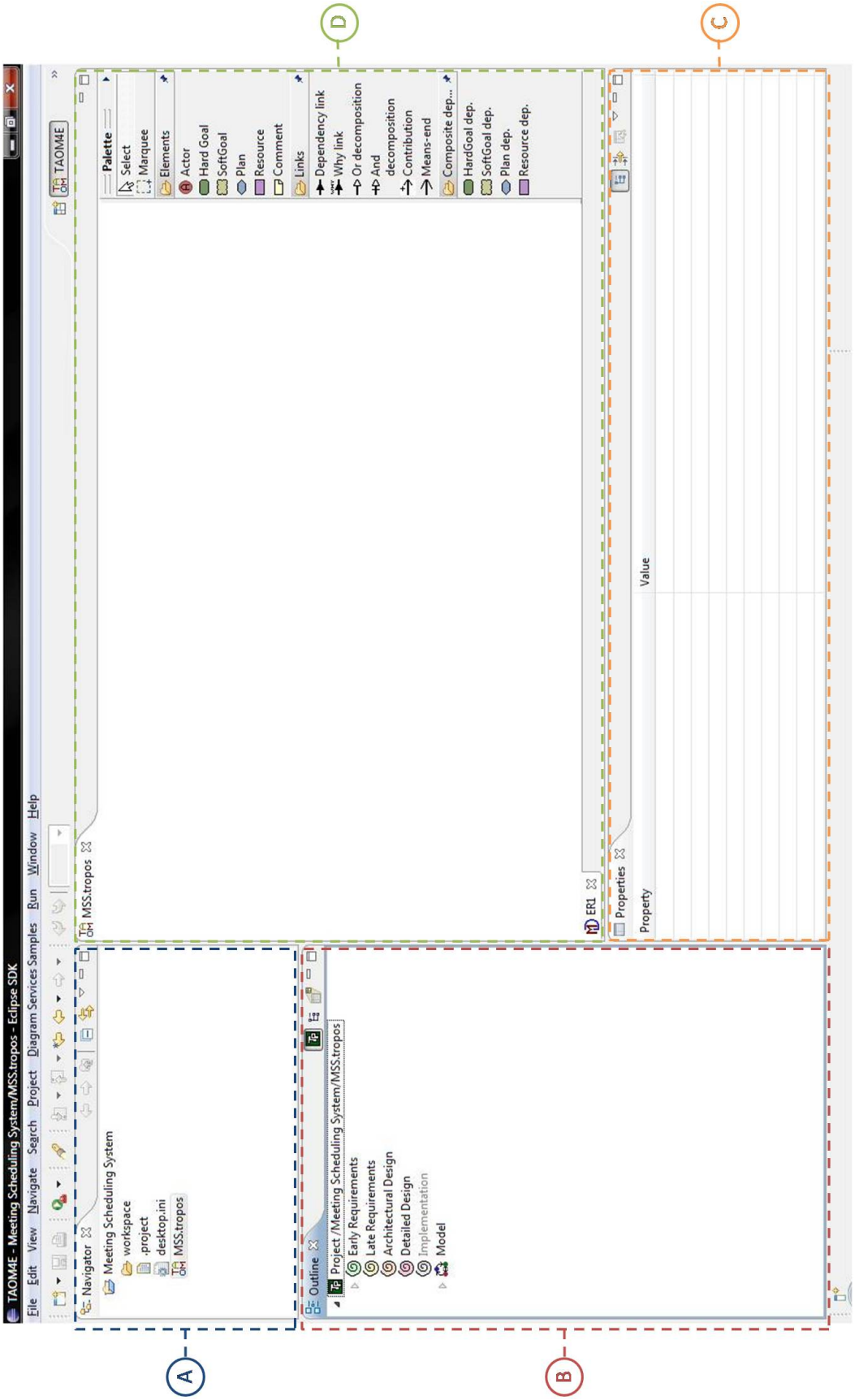


Figure 3.4: Graphical user interface of *TAOM4e*

Design and Implementation represent those phases. Moreover, one can also see diagrams associated to the different phases. So, for example in Figure 3.5, one can see that there is one Early Requirements diagram: Early requirements diagram. There is also one Late Requirements diagram: Late requirements diagram.

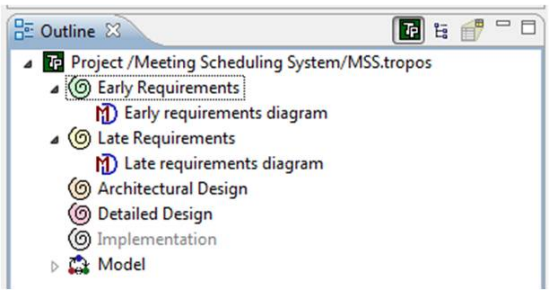


Figure 3.5: Outline tab of *TAOM4e*

The *Properties* tab (Part C of Figure 3.4) gives the properties of the element currently selected. This element can be a project, a diagram, a *Tropos* construct, *etc.* For example, Figure 3.6 depicts the properties of the Meeting initiator actor. There, one is able to set different properties for this actor. So, for example, with the actor type attribute, one can define the actor as an *Actor*, *Agent*, *Role* or *Position*. The isSystem boolean attribute defines if the actor is part of the *system-to-be* or not, *etc.*

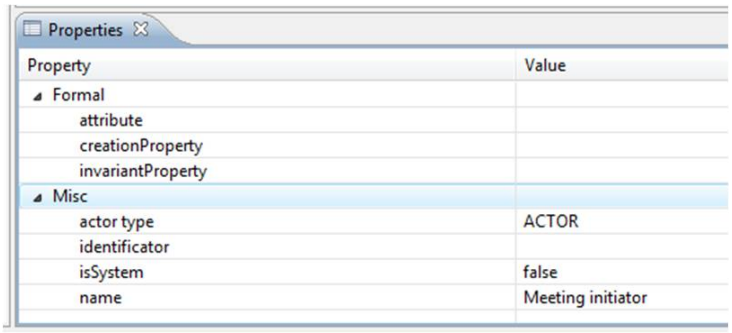


Figure 3.6: Properties tab of *TAOM4e* for the Meeting initiator actor

Finally, Part D of Figure 3.4, contains diagrams. Figure 3.7 is an example of this part of the interface with a *Tropos* diagram. At the bottom of this figure, there are tabs indicating which diagrams of the *Tropos* project are currently opened. In our case, there are two diagrams: Early requirements diagram and Late requirements diagram. It is this last which is currently visible. On the right of the figure, one can see the *Palette* of *TAOM4e*. With this part of the tool, one is able to create all *Tropos* constructs presented in the previous chapter.

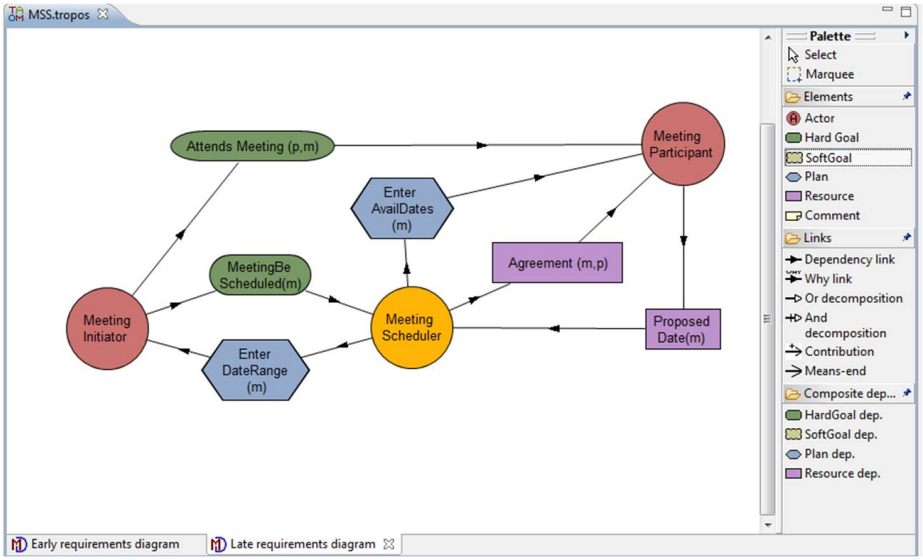


Figure 3.7: Diagram tab of *TAOM4e*

We also have to mention that one is able to change the visual properties of the different constructs. In order to do this, one has to right click on the element one wants to modify and select the Properties line. Then, if the selected element isn't a link, the window presented in Figure 3.8 appears. With this window, one is able to change the size, position, background & foreground colours, *etc.* The window for connection links is different but also allows one to change visual properties of this type of construct like the line style, width, colour, *etc.*

3.5 Summary

In this chapter, we introduced *TAOM4e* which is a tool supporting *Tropos* model development. We first gave general information about this tool. Then, we explained its architecture and, finally, we explained different elements of its graphical user interface. However, we presented only elements which are used in the next chapters.

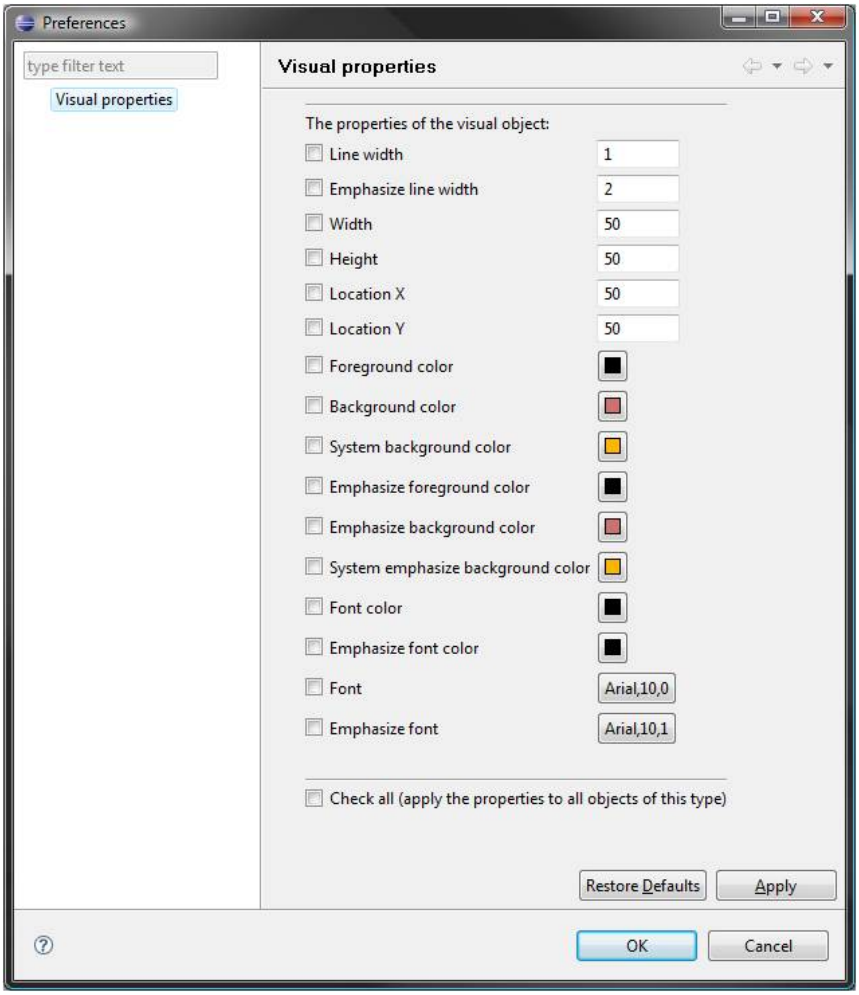


Figure 3.8: Visual properties window in *TAOM4e*

Chapter 4

Principles of graphical communication

In [21], Moody presents 9 principles which helps to build "good" diagrams. They have to be clear in reader's mind because are used along all the next chapters.

A *"good"* diagram is defined as "one which communicates effectively" [21]. The *communication (or cognitive) effectiveness* of a diagram depends on the speed, accuracy and ease required to understand the information presented in a diagram [21]. It implies that a diagram is, according to Moody, better than another if its content can be faster, more accurately and more easily understood.

Moody also notices that most IS diagrams "act as a barrier rather than an aid to user-developer communication" [21]. Consequently, the objective of his paper is to give some principles which could improve this communication. And he does this with 9 principles. In the first section, we give general information about graphical communication. The 9 next sections are dedicated to the 9 principles. Finally, in Section 4.11, we give the major conclusions of analysis made by some authors on various languages.

4.1 Graphical communication

In order to explain the principles for effective diagrams we have to introduce two elements of the theory of graphical communication.

The first one is a set of 8 visual variables. Those visual variables are depicted in Figure 4.1. Visual variables are divided in two categories: Planar variables and Retinal variables. Bertin pretends that one can combine those visual variables in order to encode the information in a graphical way [3]. However, in IS Diagrams, only one of those variables is generally used: the shape.

The second element that one has to keep in mind for the different principles presented above is the human processing of information as it is presented in [21]. It is visible in Figure 4.2. This process is composed of four different steps. The first one is the Perceptual discrimination. During this phase, the different

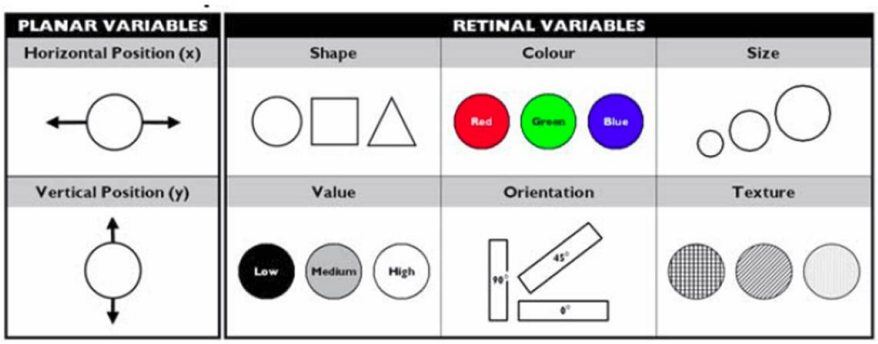


Figure 4.1: Visual variables (from [21])

elements are differentiated by the values of their visual variables. Then, during the Perceptual configuration, the information captured in the previous step is organised in groups. This information is transferred in the Working memory for active processing. The Working memory is a fast, temporary storage area which has limited capacity. The information is then transferred to the Long term memory which has unlimited storage area but is slower. The Working memory enables one to quickly analyse the information perceived by the eyes while the Long term memory allows one to store this information.

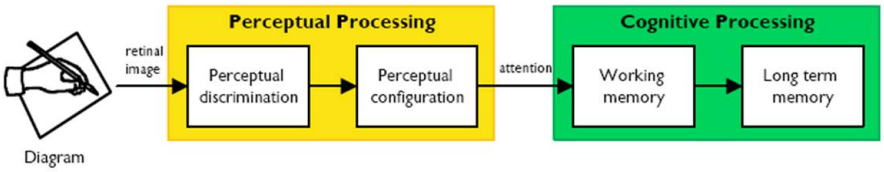


Figure 4.2: Model of graphical information processing (from [21])

4.2 Discriminability

The discriminability is related to "the ease with which diagram elements can be differentiated from the background and from each other" [21]. This parameter is important as it is the entry point of the graphical information process presented in the previous section. There are two types of discriminability: absolute and relative.

4.2.1 Absolute discriminability

The absolute discriminability is defined as "the ability to see diagram elements and separate them from the background" [21]. Moreover this type of discriminability depends on three primary factors: size, contrast and proximity.

The size of an element is a parameter determining if the reader is able to see it. Consequently, constructs must have a certain minimum size in order to be recognised (see Figure 4.3(a)). The processing of an element is optimal when its retinal image reaches 1/8 to 1/3 degrees of visual angle. So, for example, if we consider an A4 sheet of paper and a reading-distance of 25 cm, elements must have a size between 1.7 and 4 cm [21].

The contrast is the second factor. It means that the background and the different constructs should have the greatest contrast possible. So, one could, for example, use a dark-coloured background and light-coloured constructs (see Figure 4.3(b)), and conversely. This way, there is a bigger difference between elements and background. One could also use other surface properties like texture or value to differentiate elements from background.

Finally, the proximity of elements also influences the absolute discriminability. If elements are close from each others, one will require more attention to discern them (see Figure 4.3(c)). In order to avoid this drawback, one should respect a certain minimum distance between constructs. According to Moody, this value depends on the type of diagram. However, he considers that the optimal distance for most diagrams is 1-1$\frac{1}{2}$ element widths.

4.2.2 Relative discriminability

Relative discriminability is "the ability to differentiate between different types of diagram elements" [21]. The perceptual variation between the different constructs should thus be as great as possible. If the different types of constructs differ from a subtle difference, one won't probably notice it.

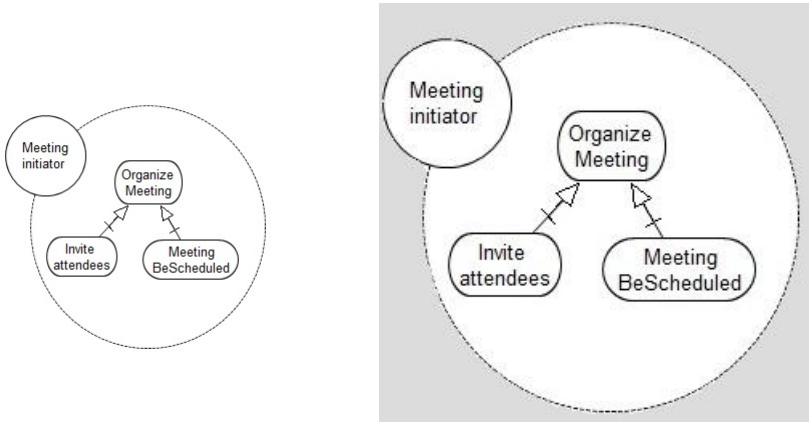
In order to improve this absolute discriminability, Moody advises to use clearly distinguishable shapes and lines for the different constructs and relationships. One can use, among other things, five basic geometric signs which are unlikely to be confused with each other: square, triangle, circle, cross and arrow. Another possibility is to use multiple visual variables. By using several variables, one increases the discriminability between construct types.

4.3 Modularity

The next principle is related to the number of constructs presented on a single diagram. Generally, modellers put a lot of information on a single diagram. Consequently, diagrams are difficult to read. Human mind has some difficulties to understand such diagrams as it has perceptual and cognitive limits.

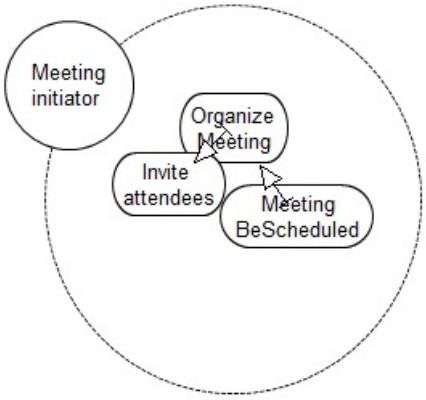
Perceptual limits are related to the number and the proximity of constructs on a single diagram. It means that if there are more elements and that they are too close from each others, the ability to discriminate them decreases.

Cognitive limits refer to the number of diagram elements which can be understood at a time. This number is limited by the Working memory which acts as a bottleneck. Miller pretends that one is able to understand "seven plus or



(a) Small elements size

(b) Contrasted elements and background



(c) High elements proximity

Figure 4.3: Absolute discriminability

minus two" concepts at a time, depending on the person [18].

A solution to this problem of diagrams containing too much information is to divide them into smaller subsystems or modules. The number of elements present in each of those subdiagrams should respect the "seven plus or minus two" rule [18]. It implies that each module should contain 5 to 9 constructs. Those numbers can seem small to modellers, but it is adapted to novices who aren't used to the diagram notation.

4.4 Emphasis

Generally, in diagrams, some elements are more important than others. However all elements often look the same. It implies that one doesn't understand at the first sight that an element is more important than others. Moreover, the most important elements could be used as an entry point of the diagram. It means that readers start reading important constructs and navigate to others

from those ones.

Moody advises to use visual variables to create a precedence among the different constructs presented on a diagram. He considers that most important elements should be highlighted while background elements should be deemphasized. This way, one knows that one has to focus on highlighted elements and that one can use them as entry points of the diagram. For example, in Figure 4.4, Meeting Be Scheduled is highlighted with a thicker border and a bold label.

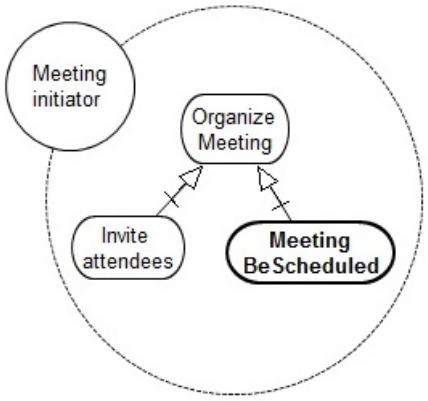


Figure 4.4: Highlighted element

4.5 Cognitive integration

This principle is related to the fact that there can be several diagrams associated with a single project. Moreover, there can be several types of diagrams, each one containing, in turn, several diagrams. This problem still becomes more pronounced with the principle which advises to divide diagrams in smaller chunks. One can have some difficulties to understand the links among diagrams and navigate through their network. It means that one requires more attention to integrate information from different sources and know where one is in the complete network of diagrams. Actually, there are two types of integration: cognitive and perceptual.

The perceptual integration relies on perceptual cues which could help to navigate among diagrams. Two specific techniques which could improve this integration type are navigational maps and signposting [21]. A navigational map is "a map showing the entire network of diagrams and the navigational path between them" [21]. It helps the user to know where s/he is in the complex network of diagrams. It also indicates her/him where s/he can go from the current diagram. Signposting helps one to know the different possible transitions from one's current position. Such navigation cues should be visible on each diagram.

The conceptual integration refers to the integration of information coming from different diagrams. Two techniques can support this kind of integration: summary diagram and current context [21]. A summary diagram is a diagram which contains a summary of the content of all diagrams of the system. This way, one has a big picture of the system. The current context technique helps one to know where one is in the system of diagrams. This information should, according to Moody, be visible on each diagram with, for example, a level numbering or a locator map (which is a reduced navigational map with a geometric shape showing where the current diagram is situated).

4.6 Perceptual directness

Perceptual directness relies on representations of elements which are spontaneous. It means that their meaning can be extracted directly by the Perceptual processing step of the graphical information processing. This mechanism reduces the effort required to understand elements signification. Such perceptually direct representations should "share properties with the objects or relationships they represent" [21]. This way there is a link between the diagram and the represented world.

Two artifacts provide perceptually direct representations. The first one is the usage of icons. Geometrical shapes generally used in IS diagrams don't convey information about represented elements because they are conventional [21]. One has to learn their meaning and keep it in the working memory. So, the advice is to use icons which resemble to represented objects. This way, one can understand their meaning without explanation or legend.

The other technique proposed is to use *perceptually direct relationships*. Indeed, the position of diagram elements can help one to understand the link among them. For example, a left to right arrangement suggests the sequence while elements situated inside others suggest the membership. However most IS diagrams don't use this property.

4.7 Structure

In most IS diagrams, there is no way to determine the structure or the grouping of elements. So, one requires more concentration to understand the information and structure it oneself. There are a lot of Gestalt laws describing how information is structured by human mind [18]. However, 3 of them are explained in [21]:

- **Proximity :** elements close from each others tend to be grouped together. For example, in Figure 4.5, Schedule meeting, Find agreeable slot, Get preferred dates and Get exclusion dates plans are situated close from each others.

- **Similarity :** similar elements tend to be grouped together. In Figure 4.5, Schedule meeting, Find agreeable slot, Get preferred dates and Get exclusion dates plans have the same visual properties.

- **Common region :** elements included in an enclosed region tend to be grouped together. In Figure 4.5, the different plans are situated in an enclosed region delimited by a dotted-line rectangle.

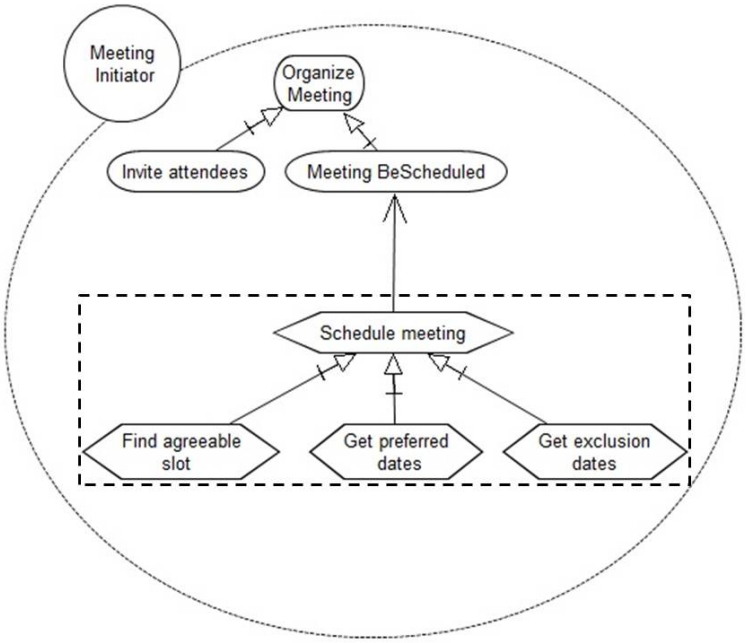


Figure 4.5: Common region

Consequently, those 3 variables should be used in order to help the reader to structure the information conveyed by a diagram.

4.8 Identification

This principle is related to the identification which is composed of two aspects: external identification and internal identification.

The external identification defines the link between the diagram and the represented world. In IS diagrams, this aspect is related to diagrams name and type. Consequently, one should give a name which defines the aspect of the represented domain (see Figure 4.6). Moreover, one should be able to know that it actually represents the title of the diagram. The type of the diagram is also important to reduce the likelihood of misinterpretation.

The labelling of diagram elements is another aspect of external identification. Indeed, in most IS diagrams, one has the opportunity to set labels to the different constructs. One should use those labels as they define the correspondence between the diagram construct and the represented element.

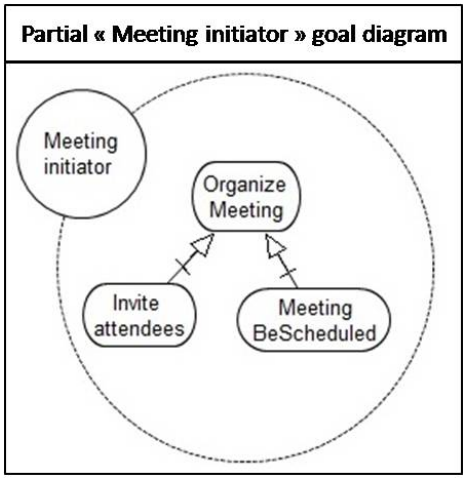


Figure 4.6: Diagram with title

On the other hand, the internal identification defines the correspondence between graphical conventions and their meaning. So, a legend or key should be visible on each diagram. It shouldn't be put on a separate sheet of paper in order to avoid cross-referencing. This legend or key can help novices. Indeed, the legend provide them an "external memory" allowing them to reduce cognitive load.

4.9 Visual expressiveness

This principle depends on the number of different visual variables used to encode information. If one wants to use multiple, parallel channels of communication, one must use multiple visual variables. Indeed, multiple visual variables conveying the same information "improve accuracy of communication and counteracts noise" [21].

As visual variables are used to convey information, one has to check that they don't convey unintended information related to perceptual and cultural associations of colours, shapes, *etc.* For this reason, all variables should be normalised. It means that:

- All diagram elements of the same type should be the same size.
- All lines of the same type should be the same thickness and style.
- All labels should use the same typeface, font size, capitalisation and weight.
- Elements should be evenly spaced to avoid unintentional grouping.
- Elements should be vertically and horizontally aligned to avoid unintentional emphasis.

However, this normalisation should be avoided in case of discrimination, emphasis or grouping.

4.10 Graphic simplicity

By opposition, graphical complexity can be defined as the number of different graphical conventions used in a notation. Notation designers can combine visual variables in order to build an almost unlimited number of constructs. However, human mind has cognitive limits on the number of visual categories that can be recognised. Moody, on basis of experimental studies, pretends that this span of absolute judgement, which is "the ability to discriminate between perceptually distinct alternatives" [21], is around 6 categories. So IS diagrams shouldn't have more than 6 different categories of constructs. But, they generally have more constructs.

One should "increase the number of perceptual dimensions on which stimuli differ" [21]. Indeed, if the number of visual variables is increased, the span of absolute judgement is improved. Another technique is also proposed: not represent everything in a graphical form. Everything can't be expressed graphically and some information can, for example, be represented in a textual form.

4.11 Application of principles for effective communication

As those principles are presented for improving the cognitive effectiveness of IS diagrams, one is able to apply them on all diagram of this domain.

So, for example, Moody considers that the relative discriminability of Entity-Relationship models [7] is very low because symbols used are similar [21]. One should thus differentiate them by using visual variables like colour.

One can also apply the principles on UML Class diagrams [20]. In this language, the relationships are the same, there is no help for managing the complexity of diagrams, only shape and texture are used as visual variables and there are 12 graphical conventions.

A deeper analysis of the cognitive effectiveness of *KAOS* [31] was carried out in [17]. The authors analysed this goal modelling language with its supporting tool, *Objectiver*. One of their conclusion is that the language can be substantially improved *wrt* all Moody's principles. So, for example, *KAOS/Objectiver* doesn't constraint the contrast between background and constructs, there is no constraint *wrt* elements proximity, model structuration depends on the modeller, *etc.*

In this document, we apply the principles for effective communication on another language, *Tropos*, and its supporting tool *TAOM4e*.

4.12 Summary

In this chapter, we presented the principles of effective communication. Next we will apply those principles to analyse *Tropos* and *TAOM4e*.

Part II

Contribution

Chapter 5

Tropos / TAOM4e analysis

Tropos, *TAOM4e* and the 9 principles for effective communication [21] have been presented in the 3 previous chapters. The goal of this second part is to combine those 3 parts by analysing *Tropos / TAOM4e* with principles for effective communication presented in Chapter 4. The inputs of the analysis carried out in this chapter are the *Meeting Scheduling System* example, the principles for effective communication, *Tropos* and *TAOM4e*. While the outputs are three recommendation lists for language engineers, tool developers and *Tropos* modellers. This process is illustrated in Figure 5.1.

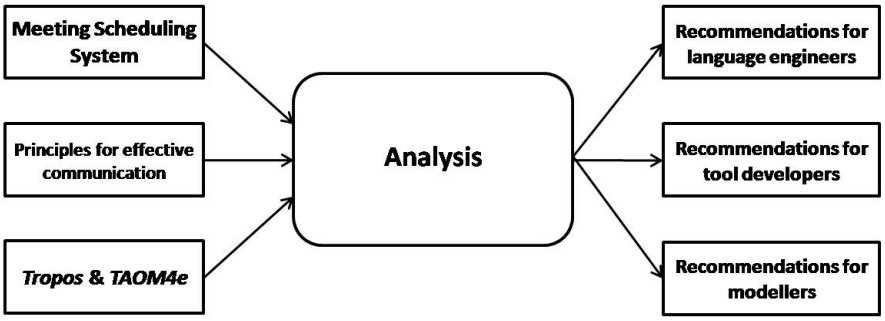


Figure 5.1: Research method for analysing *Tropos / TAOM4e*

In this chapter, we present an analysis of *Tropos* supported by *TAOM4e*. In Chapters 6, 7, 8, we discuss recommendations corresponding to language engineers, tool developers and modellers.

5.1 Elements discrimination

5.1.1 Absolute discriminability

1. Size
   By default, the size of an element is the size of its label (Figure 5.2(a)). It implies that when a modeller creates a *Tropos* construct, there is a (lot of) chance that it has a different size from the other elements of the same type. The modeller also has the opportunity to increase (Figure 5.2(b)) or decrease this size according to her/his preferences (which can be opposite to principles of "good" visualization). It means that the size of an element can be too small to be seen and recognized correctly by the reader. However, in *TAOM4e*, the minimal size of an element is limited by its label.

(a) Default size

(b) Increased size

Figure 5.2: Size of elements

2. Contrast
   In *TAOM4e*, the default colour of the background is white and can't be changed. This situation is perfect for black & white printing but can be tedious in some situations. Indeed, it requires no ink to print the background colour. In *TAOM4e*, the different elements have default colours which are visible in Table 5.1 but modellers have the opportunity to change those colours. Consequently, a risk exists that the modeller uses the same colour for elements and the background. In black & white diagrams with white constructs and background, the worst case occurs when the borders of elements are set to white because it is impossible to determine the bounds of the constructs.

3. Proximity
   In *TAOM4e*, the modeller also has the opportunity to put the elements where s/he thinks they best fit (according to personal preferences, space constraints, *etc.*). The risk is that elements have been put too close from each others. In that case, it is difficult for the reader to have a good and direct understanding of the diagram.

5.1.2 Relative discriminability

As one can see in Table 5.1, there are three different types of geometric signs: squares, circles and arrows. Elements situated in the same box of this table are

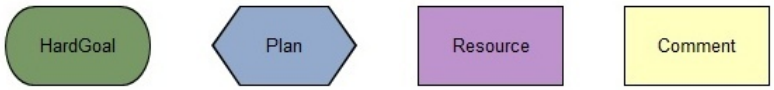
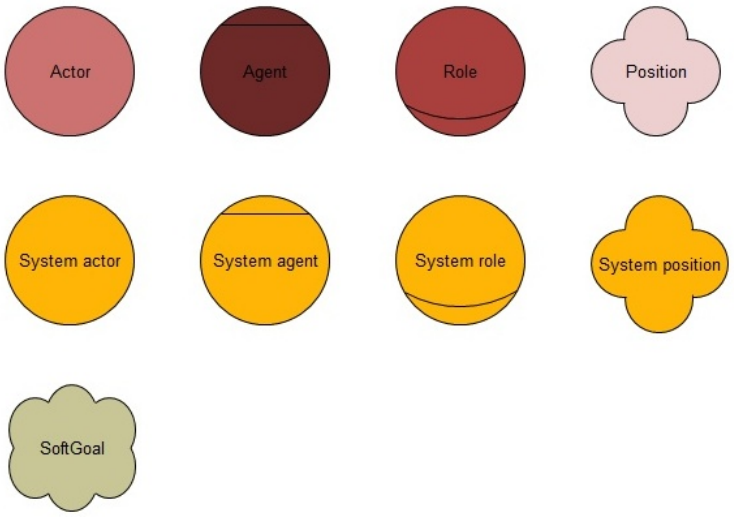
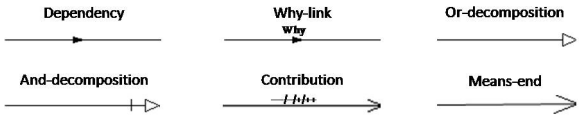
more likely to be confused with each other because of their shape.

An advantage of square constructs of *Tropos / TAOM4e* is that they have different colours by default. However, the modeller has the possibility to change them and, consequently, use the same colour.

System *Actors*, *Agents*, *Roles* and *Positions* have, by default, the same colour which can be confusing for the reader. Moreover, system *Actor*, *Agent* and *Role* constructs have the same shape and the difference among them (a line at the top or the bottom) is very subtle. The system *Position* construct has the same yellow colour as *Actor*, *Agent* and *Role* constructs but its geometric shape is quite different. Using different colours would improve the understandability.

As one can see in the last row of Table 5.1, the colour of all arrows is black. Moreover, the *and-* and *or-decompositions* have the same extremity symbol: a triangle. The single difference is a small line added to the *and-decomposition* link. The difference between *dependency* and *why* links and between *contribution* and *means-end* links are also very subtle.

Table 5.1: Construct variation

Square	
Circle	
Arrow	

5.2 Diagrams decomposition

In *Tropos*, the *"holds" relationship* can help to reduce the size of diagrams. Indeed, this mechanism allows one to close an *Actor*, *Agent*, *Role* or *Position*. If one double clicks on a closed *"holds" relationship*, then it will be expanded, and conversely. This way one is able to show the information interesting one's current interlocutor. One can, with this mechanism, focus on interactions among actors, on a single actor, *etc.*

However, this mechanism isn't sufficient. *TAOM4e* doesn't provide a mechanism which helps the modeller to divide her/his diagrams in smaller subdiagrams. Moreover, there is no mechanism checking the number of constructs on a diagram. Consequently, in *TAOM4e*, this number of elements isn't limited and one is able to put as much elements as one wants. As *TAOM4e* currently lacks mechanisms helping to divide diagrams and limiting the number of constructs, modellers are more encline to build huge diagrams.

5.3 Highlighting elements

In *TAOM4e*, there is no automatic way to put the emphasis on elements which are more important. However, the modeller has the possibility to manually change the colour, border thickness and the other visual properties in order to highlight important elements. Those properties are Line width, Width, Height, Background colour, *etc.* It implies that the highlighting of elements completely depends on the creativity of the modeller. Currently, modellers are not assisted when they want to highlight an element: they have to define themselves a different colour, a thicker border, *etc.* It implies that each modeller will probably have different highlighting methods. This can be disturbing for readers who work with different modellers. Indeed, once they are used to an highlighting method, they have to change this habit if they work with other modellers who use different highlighting conventions.

In order to be complete, we have to mention the fact that there can be involuntary highlighting. Indeed, as it was explained in Section 5.1.1, the default size of a construct is determined by its label. It implies that elements of the same type will probably have different sizes. Some will be smaller and will be interpreted as less important than other constructs of the same type with a bigger size because of their label [21]. This phenomenon is a kind of involuntary highlighting.

5.4 Diagrams integration

In *Tropos*, there are four different types of diagrams: *Early requirements*, *Late requirements*, *Architectural design* and *Detailed design*. So, stakeholders must know the role of each one. For example, *Early requirements* diagrams represent the current situation, *Late requirements* diagrams represent the domain including the *system-to-be*, *etc.* The single artifact of *TAOM4e* helping to see the order of those different phases is the *Outline tab* of *TAOM4e* (Figure 5.3). With this tab, one can know in which *Tropos* phase the diagram one is

currently reading is situated. In Figure 5.3, the current diagram is 2. Model with computer-based scheduler and its type is *Early requirements*. Note that the *Outline tab* of *TAOM4e* isn't visible on printed and exported versions of diagrams. It implies that one doesn't know to which *Tropos* phase the exported or printed diagram one is currently reading belongs.

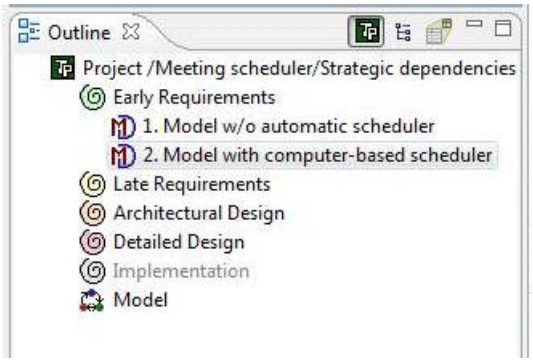


Figure 5.3: *TAOM4e* outline tab

One can also have several diagrams in a single *Tropos* phase. In our example of Figure 5.3, we have two *Early requirements* diagrams: 1. Model w/o automatic scheduler and 2. Model with computer-based scheduler. *TAOM4e* doesn't provide a mechanism which helps the reader to understand the links and to navigate among the different diagrams. So, in our example of Figure 5.3, one doesn't know the link between diagrams 1. Model w/o automatic scheduler and 2. Model with computer-based scheduler.

One can also think about another cognitive integration mechanism: an index of elements present in diagrams. As it is visible in Figure 5.4, all *Tropos* constructs of the current diagram are listed in the *Outline tab* of *TAOM4e*. In this example, there are two visible actors: Meeting Participant and Meeting Scheduler which is declared as *system*. Then, in the *balloon* (equivalent to *"holds" relationship*) of the first actor, we have all constructs which are related to its internal activity. There are also other constructs at the same level as the *balloon*: links. Actually, they are inter-actors links. Those links can, in turn, also have children: the depender or dependee (depending if it is an in or out link) and a dependum. Note that each element is characterized by the icon of the *Palette*, the name of the construct and its characteristics are also visible in the *Properties tab*.

The problem of the *Outline tab* is that it shows only the constructs of the current diagram. Consequently, one can not know automatically if a given element is present in another diagram. If one wants to get this information, one has to search manually in each diagram.

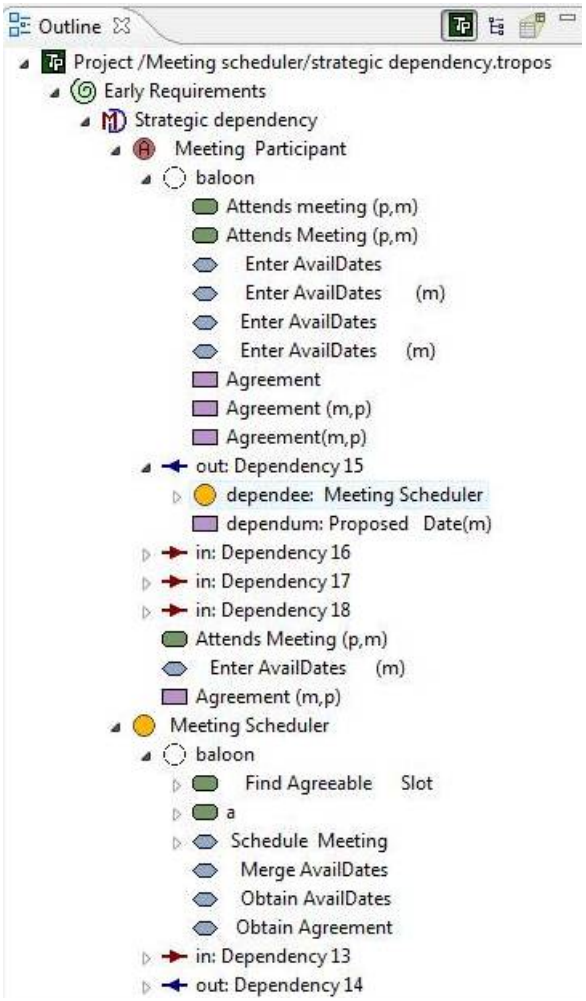


Figure 5.4: *TAOM4e outline tab* as an index

5.5 Perceptually direct representations

5.5.1 Iconic representation

Tropos is a software development methodology [6] which can be used for different types of projects in completely different domains. Consequently, its notation must be as general as possible in order to be used in completely different domains. It is consequently impossible to define icons which exactly represent modelled objects. For example, if one introduces icons which are linked to the bank domain in the *Tropos* notation, one won't be able to use *Tropos* in other domains like hospitals, cars, *etc.* Icons are strongly linked to the domain and should be modified on a case by case basis. There is a dilemma between perceptual directness and abstraction. It means that, if one introduces icons, there is less abstraction because icons are linked to a specific domain. If one

wants to keep the language as abstract as possible, one can't introduce icons and, in this case, the perceptual directness isn't improved.

5.5.2 Perceptually direct relationships

In the case of *Tropos*, a single perceptually direct relationship artifact is available: the *"holds" relationship*. This construct expresses the fact that all element inside the border of this relationship belong to a specific actor. For example, in Figure 5.5, the *"holds" relationship* of the Meeting scheduler actor is represented by the dotted circle. This artifact encloses two plans: Schedule Meeting and Merge AvailDates. This way, one knows, by the semantic of this construct, that both plans are owned by the Meeting scheduler.

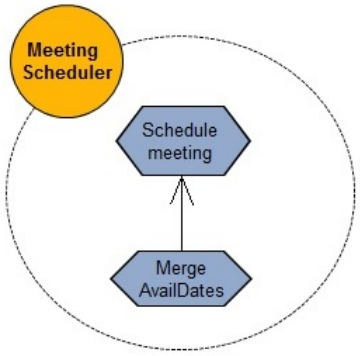


Figure 5.5: *"Holds" relationship*

5.6 Elements grouping

As it was explained in Chapter 4, the grouping of elements is related to proximity, similarity and common region.

The proximity of elements is completely determined by the modeller. S/he has the possibility to put the elements where s/he wants.

The similarity is related to the visual properties. In *Tropos / TAOM4e* elements of the same type have the same colour. As the colour is, by default, the same for all constructs of the same type, they will be grouped together by readers [21]. However, modellers have the opportunity to change the visual properties of the different constructs. It implies that modellers can do whatever they want and, for example, use different colours for all elements of a same type, same colour for elements of different types, *etc.* If modellers do so, then readers won't be able to use the colour to group elements which belong to a same group.

Tropos provides a mechanism which allows one to delimit regions: the *"holds" relationship*. It represents the bounds of an actor and all elements

situated inside the circle are linked to it. Otherwise, there is no other artifact for grouping elements together.

5.7 Diagrams correspondences

5.7.1 External identification

As it is explained in [21], name and type are part of the external identification. Labelling of diagram elements is another part of the external identification.

In the case of *Tropos / TAOM4e*, the modeller has the possibility to give a name to the diagram. This value is visible in the *Outline tab* of the tool. In Figure 5.3, we can see that there are two *Early requirements* diagrams: 1. Model w/o automatic scheduler and 2. Model with computer-based scheduler. The current situation has two drawbacks. The first is that it is not mandatory to introduce a name to a diagram. If a modeller doesn't introduce a name to a diagram, a default value is used: ER# for *Early requirements* diagrams, LR# for *Late requirements* ones, *etc.* This default value doesn't help one in understanding the link between the diagram and the represented world. The second drawback is that the *Outline tab* isn't visible on printed and exported diagrams. It implies that one doesn't know neither the name of the diagram nor its type.

Another important element for the external identification is the labeling of diagram elements. This parameter depends on the modeller. The tool doesn't have the possibility to check if the names introduced (if any) correspond to the represented world.

5.7.2 Internal identification

The internal identification corresponds to a legend or a key [21].

The *Palette* of *TAOM4e* could be considered as a legend. It contains all *Tropos* constructs and is situated on the right side of diagrams by default. However it suffers some weaknesses.

The first one is about the *Actor*, *Agent*, *Role* and *Position* constructs. There is a single icon for the four elements and it is labelled as *Actor* construct. This can be confusing for the reader because s/he doesn't know what the 3 other constructs represent if s/he only refers to the *Palette* of *TAOM4e*. Moreover, this part of the tool doesn't represent the fact that colours of the group of actor constructs vary if they are defined as system actors.

A second weakness is the fact that, if modellers use the offered opportunity to change colours, it won't be reflected into the *Palette*. This part of the tool is static, in other words it isn't updated according to visual properties.

A last weakness appears when diagrams are printed or exported: the *Palette* doesn't appear. This is quite annoying for something which could be used as a legend.

5.8 *Tropos* constructs expressiveness

In the case of *Tropos / TAOM4e*, visual variables used in diagrams are:

- Shape : As it was illustrated with the three different lines of Table 5.1, there are three categories of constructs: squares, circles and arrows.
- Colour : In Table 5.1, nearly all constructs have different colours.
- Size : One is able to use the size to discriminate elements.
- Horizontal & vertical positions : The position of an element influences the way it will be grouped.

Another part of the visual expressiveness is the "normalisation" of diagrams. In the case of *Tropos / TAOM4e*, the three first rules (all diagram elements of the same type should be the same size, all lines of the same type should be the same thickness & style and all labels should use the same typeface, font size, capitalisation and weight) are respected by the default values. However the modeller has the possibility to change them. It implies that s/he can do whatever s/he wants, including setting different visual properties for each construct. In that case, there is no more normalisation. Moreover there is no mechanism which spaces elements evenly and align them vertically and horizontally.

5.9 Number of *Tropos* constructs

Tropos seems quite complex *wrt* graphical simplicity. There are 18 graphical constructs if we consider the 4 contribution links separately, the 4 modes of actors, goals, plans, tasks, why-links, and the modes of dependencies. This number is clearly bigger than the *span of absolute judgement* which is situated, according to experimental studies, around 6 categories [18]. A positive point is that the user can add textual comments by using the notation defined for it in *TAOM4e*. It allows one to represent information which is more effectively represented in textual than in graphical mode.

5.10 Summary

In this chapter we analysed *Tropos / TAOM4e wrt* principles for effective communication. Here are the major conclusions. The discriminability principle is respected by default values but also depends on modellers. The *"holds" relationship* is the single artifact which contributes to the modularity principle. The emphasis depends on modellers. In *TAOM4e*, the *Outline tab* helps one to integrate different diagrams. The *"holds" relationship* helps one to structure the information and to have a perceptually direct understanding. The identification principle is represented by diagrams names and types which aren't visible on printed and exported diagrams. The visual expressiveness relies on 3 visual variables. Finally, we can also say that *TAOM4e* is quite complex with its 18 different constructs. Table 5.2 summarizes our analysis of *Tropos / TAOM4e*.

Table 5.2: *Tropos / TAOM4e* analysis : Summary

Principle	**Criteria**		**Evaluation**
Discriminability	Absolute	Size	Depends on the size of the label
		Contrast	Default colours respect contrast
		Proximity	No position constraint
	Relative	Shapes	Squares, circles & arrows
		Colours	4 constructs have the same colour
Modularity	Hide elements		The *"holds" relationship* helps
	Divide diagrams		No help to divide diagrams
	Number of elements		Not limited
Emphasis	Highlighting of elements		No automatic highlighting
Cognitive integration	List of diagrams		Visible in the *Outline tab*
	Links between diagrams		Not visible in *TAOM4e*
	Index of elements		The *Outline tab* lists constructs
Perceptual directness	Iconic representation		No icon
	Perceptually direct rel.		The *"holds" relationship*
Structure	Proximity		Depends on modellers
	Similarity		Depends on modellers
	Common region		The *"holds" relationship*
Identification	External	Diagram name	Depends on modellers
		Diagram type	Visible in the *Outline tab*
		Labels	Depends on modellers
	Internal	Legend	The *Palette* of *TAOM4e*
Visual expressiveness	Visual variables		Shape, colour, size & position
	Constructs normalisation		Depends on modellers
Graphic simplicity	Number of different constructs		18

Chapter 6

Recommendations for language engineers

The previous chapter presented an analysis of the visual aspects of *Tropos* (associated with *TAOM4e*) *wrt* 9 principles for producing effective diagrams [21]. This study showed up that the *Tropos* modelling language (associated with *TAOM4e*) suffers some lacks: same colour used for different constructs, no way of delimiting regions for structuring, *etc.*

In the three next chapters, we give some solutions to the previously explained weaknesses of the *Tropos* / *TAOM4e* association. We provide a summarized name of the recommendation in the title of each section. Next, each recommendation is described in detail and synthesized in the section table.

This chapter presents some recommendations for *Tropos* language engineers. Before reading the recommendations of this chapter, one must know that language engineers are people who define(d) the syntax and the semantic of a language. In our case, this language is *Tropos*. The recommendations made here will thus imply changes in the syntax and/or the semantic of the *Tropos* language.

6.1 Differentiate diagram elements using visual variables

The first recommendation, presented in Table 6.1, is related to the discriminability and, especially, the relative one. Indeed, visual variables used to represent different constructs should be clearly distinguishable from each others [21].

As we have seen in Chapter 5, *Tropos* faces some problems concerning this discriminability. Indeed, system *Actor*, *Agent* and *Role* constructs have the same shape, colour and only differ from each others in a small line as illustrated in Figure 6.1.

Table 6.1: Differentiate diagram elements

Title	LE1: Differentiate diagram elements using visual variables
Goal	Increase relative discriminability
Description	System *Actor*, *Agent* and *Role* constructs nearly have the same visual properties. They should have a visual variable discriminating them from each others

Figure 6.1: *Actor*, *Agent* & *Role* constructs

This is confusing for the user. S/he has to carefully pay attention to the slight difference between those three different actors of the modelled domain.

A solution would be to use different colours for each of those three constructs. But we have to take care of the colours of the other constructs: we can't use the same colour as e.g. *Plan*, *Resource*, *etc.* Figure 6.2 suggests a solution to the problem of the different types of actors having the same colour. Note that colours are quite similar. This property makes possible to set a difference between the different elements while the reader also understands that the shapes share some properties.

Figure 6.2: A solution to the discriminability problem

A further improvement for these constructs could be the use of another visual variable which differentiates them; for example the shape of the construct (eg. as it is done for the *Position* construct in Table 5.1). However, the disadvantage of this solution is that one looses the similarity of the different actors.

Consequently, we can consider that the solution presented in Figure 6.2 is the best one: the three constructs are slightly different by their colour and a small black line. Note that the slight difference of colour is also visible when diagrams are printed in black & white. Another visual variable could also be added to differentiate the three constructs: 3D. But we don't discuss it more deeply.

6.2 Provide a summary diagram and a navigational map in order to support diagram decomposition in manageable chunks

This recommendation, summarized in Table 6.2, addresses the cognitive integration of the user [21]. The problem is that, if there is a lot of diagrams, one can hardly guess the links between them. The goal of a summary diagram or navigational map is thus to help the reader in the diagram integration process.

Table 6.2: Support diagram decomposition

Title	LE2: Provide a summary diagram and a navigational map in order to support diagram decomposition in manageable chunks
Goal	Increase the cognitive integration
Description	Navigational maps and summary diagrams are two ways to help the user to synthesize all the information coming from different diagrams

As it was explained in Chapter 4, it is easier for the reader to read small diagrams and then understand the global idea with the links between them.

At the moment, in *Tropos*, the modeller is responsible for dividing diagrams in manageable chunks. But, if models are decomposed, readers can get lost. For this reason, a summary diagram could be useful. This diagram should contain only the main elements. A solution is a diagram showing only the actors and their dependencies without taking the elements contained into their *"holds" relationship* into account as illustrated in Figure 6.3 *actor diagram*. If the reader wants to focus on a single actor it can use its *goal diagram*.

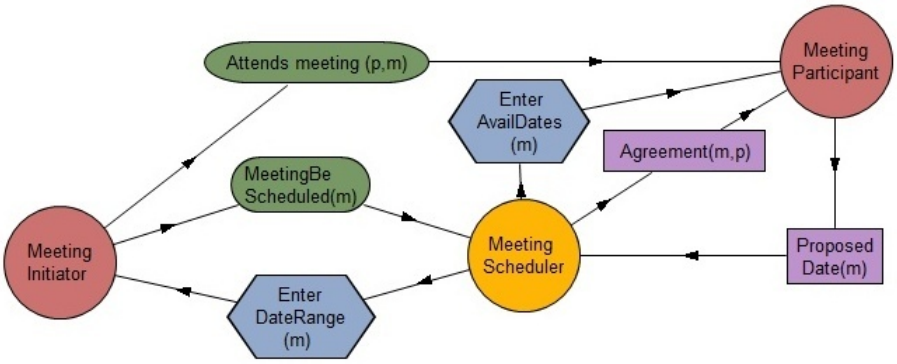


Figure 6.3: An example of summary diagram

A navigational map is also useful for readers. Such a map shows the complete network of diagrams and the different relationships among them. This way, one knows where one can go and where one is in the complete network of diagrams. This artifact is illustrated in Chapter 9.

6.3 Define navigation cues

The goal of navigation cues (see Table 6.3) is to facilitate integration of different diagrams. Moody considers that the user should be aware of all possible transitions from the current diagram [21].

Table 6.3: Navigation cues

Title	LE3: Define navigation cues
Goal	Increase the cognitive integration
Description	Navigation cues should help the reader to easily navigate among linked diagrams

Recommendation LE2 advises to add a navigational map. With this map, one is thus able to see the complete network of diagrams. Further one needs to be helped in the navigation through the different elements of this navigational map. We propose thus to introduce navigation cues associated to the navigational map. They are represented in Figure 6.4.

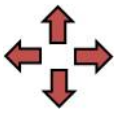
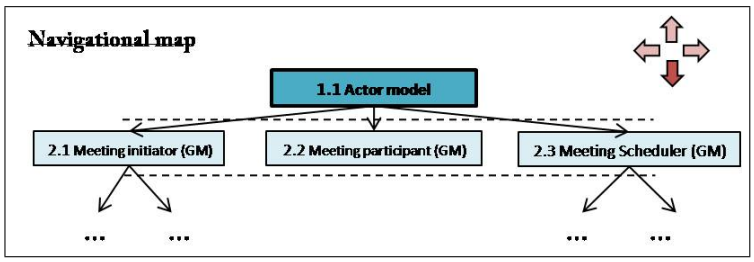


Figure 6.4: Navigation cues for navigational maps

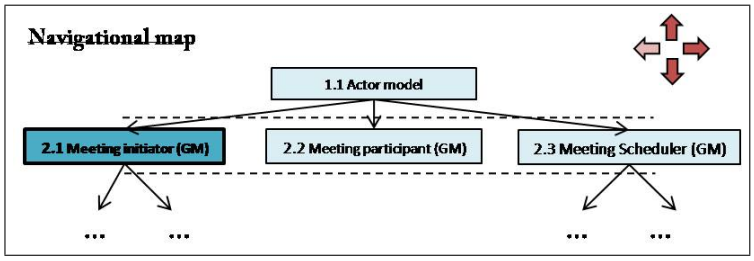
If one is currently visualising a diagram, one can see all its surrounding diagrams in the navigational map. Now, with the different arrows of Figure 6.4, one can directly reach the diagrams linked to the current one. So, for example, if one clicks on the *down* arrow, one will be redirected to a diagram situated above the current one in the navigational map. In this case, the *up* arrow redirects the reader to a father diagram, the *down* one to a lower diagram and the *left* and *right* ones to a brother diagram. So, for example, in one clicks on the *down* arrow in Figure 6.5(a), one will be redirected to the Meeting Initiator (GM) diagram as it is visible in Figure 6.5(b).

This navigation cue could also be included in the diagram itself. The result of clicking an arrow would be the same as explained for the navigational map.

This mechanism could also be applicable to the navigation through the different phases. This is represented in Figure 6.6. There, two arrows, *left* and *right*, are represented. Their role is to go backward and forward through the different steps. In our example, the current step is 3.Architectural design (visible



(a) Before transition



(b) After transition

Figure 6.5: Example of navigation cues in a navigational map

with the bold highlighting). If one clicks the *right* arrow, one will be redirected to the 4.Detailed design diagram(s). Conversely, if one clicks the *left* arrow, one will be redirected to the 2.Late requirements diagram(s).

Figure 6.6: Navigation cues for *Tropos* phases

6.4 Introduce icons in language elements to have a better direct understanding

This recommendation, described in Table 6.4, refers to the perceptual directness [21] of diagram constructs. Icons reduce cognitive load because they have built-in mnemonics.

The problem of the shapes used in *Tropos* as in most IS diagramming languages, is that they have been defined arbitrarily. They convey no information. It means that the reader has to learn their meaning. This is not a problem for modellers who are used to those conventions. Their clients, however, have to learn their signification in order to understand the model. This is time-consuming and could be confusing. The use of icons could reduce the learning

Table 6.4: Icons

Title	LE4: Introduce icons in language elements to have a better direct understanding
Goal	Increase visual expressiveness
Description	Icons refers, according to [21], to the direct comprehension of the reader by using symbols s/he knows

time and facilitate the comprehension of diagrams.

We advise to introduce an icon for *Actor* constructs, as shown in Figure 6.7. It symbolises the fact that the *Actor* construct is a human actor (by opposition with *System* actor present in *TAOM4e*). This icon seems abstract enough. Indeed, the problem of icons is that they are generally related to a specific domain. Consequently, it is difficult to introduce such artifacts in modelling languages. However, the icon chosen here is abstract enough to be used in all domains. For *Goal*, *Plan* and *Resource* constructs, it is more difficult to define such icons because those language elements are too abstract in their meaning to have an iconic representation.

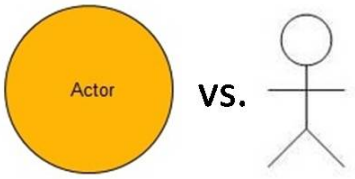


Figure 6.7: Current vs. iconic representation of an actor

6.5 Introduce techniques to group elements

The goal of this recommendation summarized in Table 6.5 is to help the reader to structure the information. We can structure elements according to three properties: proximity, similarity and common region [21].

Currently, modellers are able to group elements by using two artifacts: colour and proximity. We suggest to add a third artifact for the modeller: a way of delimiting regions. This can be a shape like a rectangle, a circle, *etc.* Currently, the *"holds" relationship* is a technique to group elements but it is limited to the "perimeter" of an actor. The idea of the proposed artifact is to make possible to group some elements "inside" an actor, belonging to several actors, *etc.* together.

Table 6.5: Grouping elements

Title	LE5: Introduce techniques to group elements
Goal	Contribute towards the grouping of elements
Description	We suggest to add a visual shape determining a closed area containing logically linked elements

6.6 Make a manual

The goal of such a manual (see Table 6.6) is to address the cognitive integration [21]. Indeed, if one knows the semantic of all elements of a notation, one is more encline to understand the diagrams, the links between the different phases, *etc.*

Table 6.6: Manuals

Title	LE6: Make a manual
Goal	Address the identification principle
Description	A manual is useful to learn/understand the principles of a language

Tropos doesn't have such a reference document. It is important to have a single complete book or document where people are sure to find what they are looking for.

6.6.1 Make different manuals for modellers and end-users

The goal of this recommendation expressed in Table 6.7 is to have manuals adapted to their readers. In other words, it means that the objective is to give the appropriate information to the right person in order to facilitate her/his understanding of the diagram(s).

Table 6.7: Different manuals

Title	LE6.1: Make different manuals for modellers and end-users
Goal	Address the identification principle
Description	Modellers and readers don't require the same knowledge of *Tropos*. Consequently, they should have distinct manuals

Here, we suggest to write two distinct manuals: one for modellers and one for end-users who read the model. The two categories of people don't need the same information. Modellers need a complete information about the language with, for example, the meta-model. While readers only need a part of the information

like the semantic of the different constructs, the role of the different stages, *etc.* This separation prevents from disturbing the end-user with useless information and, oppositely, to not give all the required information to modellers.

6.7 Summary

In this chapter, we proposed recommendations for language engineers based on principles for effective communication [21]. The list presented hereunder synthesizes those recommendations for language engineers.

- LE1: Differentiate diagram elements using visual variables
- LE2: Provide a summary diagram and a navigational map in order to support diagram decomposition in manageable chunks
- LE3: Define navigation cues
- LE4: Introduce icons in language elements to have a better direct understanding
- LE5: Introduce techniques to group elements
- LE6: Make a manual
  - LE6.1: Make different manuals for modellers and end-users

In the next chapter, we propose a list of recommendations for tool developers. This list is also based on our analysis of *Tropos/TAOM4e*.

Chapter 7

Recommendations for tool developers

In this chapter, we present a list of recommendations for tool developers. They are related to *TAOM4e* according to the fact that the analysis was made on basis of this single tool.

Before starting, we have to mention that tool developers are people who can be considered as the intermediary between language engineers and modellers. Their role is to produce tools usable and useful to the modellers. Their tool has to comply with the syntax and the semantic of the language, *Tropos* in our case, if they want it to be used to create models.

7.1 Pay attention to the absolute discriminability

The goal of the different recommendations presented in this section is to respect the discriminability principle [21]. Indeed, the *absolute discriminability* is a part of the discriminability principle. Hereunder, we give some recommendations linked to the size and proximity of elements and their contrast with the background.

7.1.1 Size

A first proposition summarized in Table 7.1 is to impose a minimal size for elements. Indeed, as it is explained in [21], elements must have a minimal size in order to be read. If an element is too small, one will have difficulties to read it. The goal of the proposed mechanism is thus to avoid that modellers put elements too close from each others.

A mechanism which sets the same size to all elements of a certain type (independently of their label) is, according us, another improvement (see Table 7.2). Indeed, it is easier to discriminate elements of a same type among others if they have the same size [21]. As it is explained at the beginning of this section, this recommendation is linked to the discriminability principle.

Table 7.1: Minimal size

Title	TD1.1: Minimal size
Goal	Check the size parameter of the absolute discriminability
Description	Check that the different constructs have a certain minimal size

Table 7.2: Same size for elements of the same type

Title	TD1.2: All elements have the same size
Goal	Check the size parameter of the absolute discriminability
Description	Check that all elements of the same type have the same size

7.1.2 Contrast

The modeller should be able to change the colour of the background (see Table 7.3). It could be useful when modellers need light coloured elements associated with a dark background, for example. This solution can be useful in some specific situations like including the diagram in a specific presentation template imposing specific background colours.

Table 7.3: Change background colour

Title	TD1.3: Allow to change background colour
Goal	Address the contrast parameter of the absolute discriminability
Description	Allow one to change the background colour

The tool should also pay attention to the contrast between elements and background (see Table 7.4). This recommendation is also applicable to the current version of the tool allowing only white background because modellers are able to draw white constructs. One should be able to deactivate this mechanism if one wants to print diagrams with a black & white printer. Indeed, in this case, one will probably use white background and white constructs.

7.1.3 Proximity

As it is mentioned in Section 5.1.1 modellers can put elements wherever they want. A solution to this problem would be to impose a minimal distance between elements (see Table 7.5). This way, modellers' actions are restricted and they can't produce diagrams with elements too close from each others.

Recommendation TD1.5 can seem very constraining for the modeller. Another recommendation related to the proximity summarized in Table 7.6 is a

Table 7.4: Contrast between background and constructs

Title	TD1.4: Check contrast between constructs and background
Goal	Address the contrast parameter of the absolute discriminability
Description	Check that the colours of constructs and background are different to avoid readers' confusion

Table 7.5: Minimal distance between elements

Title	TD1.5: Minimal distance between elements
Goal	Address the proximity parameter of the absolute discriminability
Description	Check that there is a minimum distance between the different constructs of a diagram

mechanism which automatically reorganizes elements. The idea here is to select some/all nodes and reorganize them on basis of a reorganization algorithm which respects a minimal distance. This mechanism can also suggest a direct interpretation. Indeed, if one has a tree layout, the position of subelements suggests that they are a decomposition of the root. The reorganization of selected elements, e.g. in a tree shape, also influences the human mind which will group the root and all the leaves together. It is due to the fact that the elements belong to a certain structure.

Table 7.6: Automatic reorganization of elements

Title	TD1.6: Automatic reorganization of elements
Goal	Address the proximity parameter of the absolute discriminability
Description	Provide a mechanism which automatically places (selected) elements of a diagram

7.2 Pay attention to the relative discriminability

The goal of the recommendations presented in this section is the same as the one presented in Section 7.1: respect the discriminability principle [21].

A weakness pointed out in Chapter 5 is that system *Actor*, *Agent*, *Role* and *Position* constructs were nearly the same. The first recommendation summarized in Table 7.7 is, consequently, quite direct: use different colours for *Actor*, *Agent*, *Role* and *Position* constructs. It will reduce the confusion rate among

the different readers. This amelioration makes possible to contribute towards the discriminability.

Table 7.7: Different colours for system constructs

Title	TD2.1: Use different colours for system actor constructs
Goal	Improve the relative discriminability
Description	The colours of system *Actor*, *Agent Role* and *Position* constructs should be different

Another proposal is to check that colours chosen by users are different for each type of element (see Table 7.8). With this solution, the different types of elements won't have the same colour and it will be easier to distinguish them from each others. Without this mechanism, elements of different types having the same colour could wrongly be grouped together.

Table 7.8: Different colours for the different types of constructs

Title	TD2.2: Check that colours are different (Inter construct types)
Goal	Improve the relative discriminability
Description	The colours of the different types of constructs (*Hadgoal*, *Plan*, *etc.*) should be different

Recommendation TD2.2 only checks if colours are different among the different types of constructs. But the tool should also verify that all elements of a certain type have the same colour (see Table 7.9). If elements of the same type have the same colour they are more encline to be grouped together [21]. However highlighted elements should be excluded from this verification because they may have a different colour.

Table 7.9: Same colour for constructs of the same type

Title	TD2.3: Check that colours are the same (Intra construct type)
Goal	Improve the relative discriminability
Description	The colour of the elements of the same type should be the same excepted in case of highlighting

7.3 Hide & show decomposition trees

The goal of this recommendation, summarized in Table 7.10, is to allow the modeller to hide some elements on a diagram in order to facilitate the comprehension of the reader. Indeed, hiding some elements implies that there are less constructs on the diagram and, consequently, according to the modularity principle [21], one will understand the diagram easier.

Table 7.10: Decomposition trees

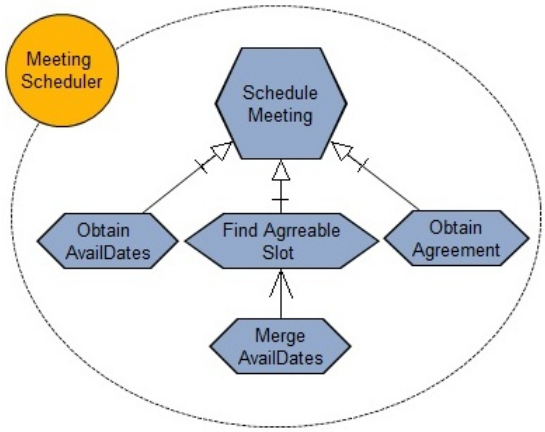
Title	TD3: Hide & show decomposition trees
Goal	Increase modularity
Description	Add a mechanism showing & hiding decomposition trees Indicate when a tree is closed Manage links with hidden elements

An example of decomposition tree is visible in Figure 7.1(a). There, the tree is composed of four constructs: Schedule Meeting, Find agreeable slot, Obtain AvailDates and Obtain Agreement. Schedule Meeting plan is the root of this tree. Indeed, it is decomposed in three subelements: Find agreeable slot, Obtain AvailDates, Obtain Agreement.

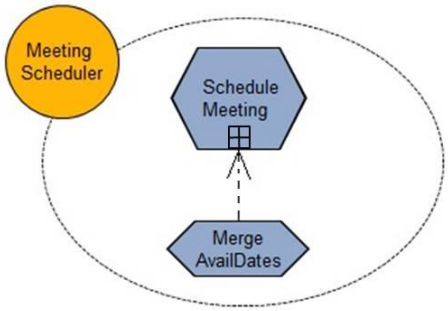
Our recommendation is to make possible to open and close such a tree. It means that when one double clicks on the root of a decomposition tree which is expanded, all its subelements will be hidden. In our example of Figure 7.1(a), if one double-clicks on the Schedule Meeting plan, all the other constructs of its decomposition tree (Find agreeable slot, Obtain AvailDates and Obtain Agreement) will be hidden. Conversely, if one double clicks on the root of a closed decomposition tree, all its subelements will be visible.

Moreover, we advise to add a symbol in the root construct when the tree is closed. We could, for example, use a + symbol on the root of the closed tree. This symbol means that this construct is the root of a decomposition tree which can be expanded. This principle is illustrated in Figure 7.1: the decomposition tree of the diagram of Figure 7.1(a) has been replaced by its root, Schedule Meeting, containing a + symbol in diagram of Figure 7.1(b). If one sees this + symbol on the Schedule Meeting construct, one knows that this construct is the root of a closed tree. Consequently one understands that it is possible to expand this tree.

However, this solution implies a new problem: there can exist relationships initiating or finishing at an hidden construct. So, we have to find a solution in order to not loose this information. If a link starts or ends at a subnode of a closed tree, one of the solutions might be to use a dotted link starting/ending at the root of the closed tree. This is illustrated in Figure 7.1. The mean-ends link between Merge AvailDates and Find Agreeable Slot from 7.1(a) has been transformed in a doted mean-ends link between Merge AvailDates and Schedule Meeting which is the root of the closed tree, in 7.1(b).



(a) Original version



(b) Proposed version

Figure 7.1: Hide a decomposition tree

7.4 Use different layers

This recommendation suggests to hide some elements in order to facilitate the comprehension of the reader. If there are less elements on a diagram, readers are more encline to understand it. This recommendation is summarized in Table 7.11.

Our recommendation is to use different layers, each one containing a part of the global model. For example, a modeller can make a diagram containing two aspects: security and maintenance. It implies that this diagram will include some actors which are *maintenance* ones and some *security* ones. When the modeller presents the diagram to the client, s/he can hide the *maintenance* constructs if s/he wants to focus on the *security* aspect. Conversely, if the reader is more interested in the *maintenance* aspect, the modeller can hide *security constructs*. This way, there are less elements on the diagram and readers can focus on aspects which interest them. However we can't illustrate it with our *Meeting Scheduling System* example.

Table 7.11: Layers

Title	TD4: Use different layers
Goal	Increase modularity
Description	Add the possibility to put constructs on different layers This solution makes possible to hide all elements of a given layer

7.5 Hide some elements

The goal of the recommendation summarized in Table 7.12 is related to the modularity principle [21]. Indeed, the fact of hiding some constructs reduces the number of elements presented on a single diagram. And users are more encline to understand a diagram if there are less constructs [21].

Table 7.12: Hiding of elements

Title	TD5: Hide some elements
Goal	Increase modularity
Description	Make possible to show only constructs current readers are interested in

The tool should thus offer the possibility to hide some elements. In other words, the reader should be able to see only elements s/he is interested in.

A solution to this situation would be to insert a *"Show/Hide"* option when a *Tropos* construct (or a group of them) is selected. This option could, for example, be present into the *right-click menu*. More general options like *Hide all Actors, Roles, Positions, etc.* would also be helpful. When an actor is hidden, all its sub-elements should also become invisible.

7.6 Provide ways of highlighting elements

The recommendation presented in Table 7.13 is related to the emphasis principle [21]. Indeed, in Chapter 5, we pointed out the fact that modellers had to emphasize elements themselves, in other words, there is no mechanism helping them to highlight elements. It implies that each modeller will probably use different highlighting techniques.

The tool should thus provide a mechanism which allows one to highlight selected elements. It means that, when a modeller selects a *Tropos* construct, an option is activated in the menu bar or somewhere else. Then, when s/he clicks this button, the visual properties of the selected element are changed according to some parameters like the current values of the different properties, *etc.* This way, modeller's work is simplified and s/he doesn't loose time in thinking about

Table 7.13: Highlighting

Title	TD6: Provide ways of highlighting elements
Goal	Help modellers to respect the emphasis principle
Description	Provide a mechanism which highlights selected elements in a standard way Allow personalization of the highlighting functionality of the tool (TD6.1)

contrast among colours, *etc.* Note that one should also be able to select some elements (not a single) in order to highlight them (also if they have different types). The important point is that it is the modeller who decides which elements have to be highlighted.

Default values for automatic highlighting will often be useful and aren't time consuming but modellers can require different parameters values for various reasons. The highlighting mechanism has default values which should thus be modifiable according to external constraints, personal preferences, *etc.* The tool should thus provide a window, similar to the *Preferences* window of *TAOM4e*, allowing one to see and set the different parameters used for the highlighting.

7.7 Provide ways of delimiting regions

Structure principle [21] explains how to group the information. We propose to provide ways of delimiting regions. The recommendation presented in Table 7.14 aims at explaining the reader how to structure the information presented on a diagram.

Table 7.14: Delimiting regions

Title	TD7: Provide ways of delimiting regions
Goal	Help readers to structure elements
Description	Introduce an artifact delimiting regions on diagrams

The tool should propose a way of delimiting regions. The idea here is to help the modeller in her/his work of putting elements in such a way that they are interpreted as logically linked by the reader. This can be done with, for example, a geometrical shape and more particularly with a dotted-line rectangle, circle, *etc.* All elements situated inside this rectangle will certainly be grouped together by the reader. In Figure 7.2, there are three groups delimited by dotted rectangles. All elements enclosed inside the red dotted rectangle are related to Meeting Initiator and Meeting Scheduler. While element in the blue rectangle are related to Meeting Scheduler and Meeting Participant. This way, one directly knows that, for example, Enter DateRange(m) has to be associated with Meeting

Initiator and Meeting Scheduler actors. We can also mention that elements inside the orange rectangle are related to Meeting Initiator and Meeting Participant.

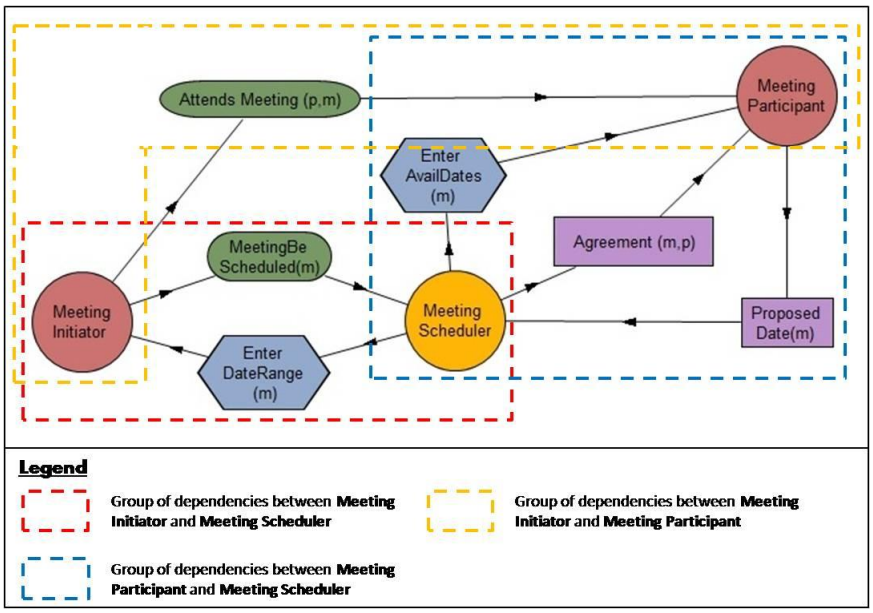


Figure 7.2: Exemple of grouping artifact

This artifact can also be useful when a construct is linked to two (ore more) completely different groups. In such a situation, it is difficult for the modeller to put the element on the diagram in such a way that it will be interpreted as appertaining to both logical groups.

Moreover, geometrical shapes make possible to define subgroups inside top-level groups. In this situation, modellers are unable to use proximity because elements of subgroups are necessarily close from all elements of the supergroup.

However, those two last situations can't be illustrated with the *Meeting Scheduling System* example.

7.8 Provide a legend

The single element which can be considered as a legend in *TAOM4e* is the *Palette*. But this part of the tool isn't visible on exported and printed diagrams. Our recommendation, summarized in Table 7.15, is to add a legend which is visible also on exported and printed diagrams. It helps the reader to understand the graphical conventions [21].

Our advice here is to add a legend. It should also be visible on exported diagrams which are included into paper reports. This would contribute towards the understanding and the dialogue with readers who are not familiar with *Tropos*.

Table 7.15: Legend

Title	TD8: Legend on each diagram
Goal	Contribute towards internal identification
Description	Add a legend on diagrams Make this legend also visible on printed and exported diagrams Update this legend according to used elements (TD8.1) Update legend *wrt* visual properties (TD8.2) Allow modellers to modify this legend (TD8.3)

This proposal could still be improved. Sometimes, modellers don't use all *Tropos* constructs. Introducing an "overloaded" legend can disturb the reader. Consequently, we advise to remove elements which aren't present in the diagram from the legend. With this solution, the reader won't loose time in searching elements which aren't present on the diagram, s/he won't ask questions.

In Section 3.4.3, we mentioned that modellers are able, in *TAOM4e*, to change colours of elements. It can be confusing for the reader if, for example, the colour of a construct is changed and this variation isn't represented in the legend. Consequently, we advise to use an updated legend. It means that the visual aspect of the icons situated in the legend is the same as the visual aspect of the elements of the model. This way, readers' work is facilitated.

Recommendations TD8, TD8.1 & TD8.2 related to the legend are illustrated in the example of Figure 7.4. This legend contains only elements present in Figure 7.3. Indeed, constructs like *Softgoals* and *Resources* aren't represented in the legend of Figure 7.4. Moreover, an attentive reader will probably notice that, compared to previous diagrams of this section, the colour of Meeting Scheduler in diagram of Figure 7.3 has been changed. This visual modification is, according to our recommendation, also visible in the legend of Figure 7.4.

A question still remains: what to do with highlighted elements which have different visual properties. The answer is quite direct and simple: don't care about it. Highlighted elements are exceptions and adding them to the legend would overload it and disturb readers.

The modeller should also be able to modify this legend. A modeller can, for example, discuss about an element which isn't still present in the legend. If s/he adds it to the legend, it can facilitate the discussion with her/his interlocutors. For example, the diagram of Figure 7.3 contains the same elements as the one Figure 6.3 excepted that the two resources, Agreement(m,p) and Proposed Date(m), have been removed in the first diagram. A possibility is that the modeller proceeds incrementally: s/he tries to build the model by first identifying the different actors, then the goal dependencies, *etc.* Imagine that Figure 7.3 represents the step just before the identification of *Resource* dependencies. If the modeller includes this *Resource* dependency in the legend, as it is done in

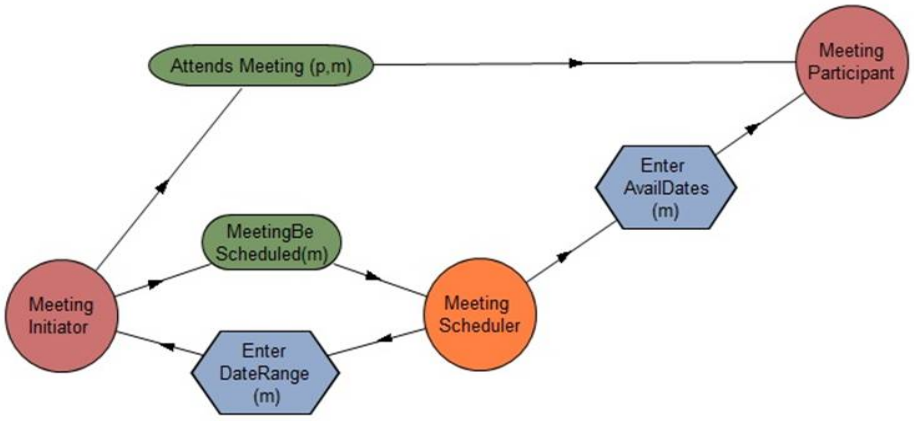


Figure 7.3: Example of diagram for a legend

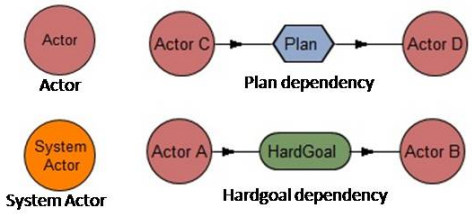


Figure 7.4: Updated legend

Figure 7.5, s/he has a reference to discuss about this type of construct with her/his interlocutor(s).

7.9 Print diagram name & type

The goal of this recommendation, summarized in Table 7.16, is related to the external identification principle [21]. The name and the type of a diagram are a link between the diagram and the represented world. But, with *TAOM4e*, as it is explained in Chapter 5, the name and the type of the diagram aren't visible on exported and printed diagrams.

Table 7.16: Print diagram name & type

Title	TD9: Print diagram name & type
Goal	Contribute towards external identification
Description	Print diagrams names and types when they are exported/printed

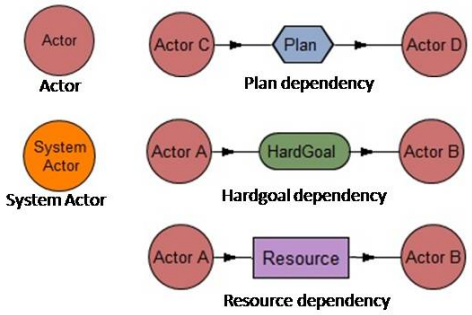


Figure 7.5: Legend with extra constructs

Consequently, our recommendation is simple: include types and names on exported and printed diagrams. It will facilitate readers' work, reduce errors rate, *etc.* A modeller can, for example, send a set of diagrams which are slightly different to a printer. When s/he takes the different sheets, s/he will easily recognize the different versions with the name and in some cases the type printed on the diagram. But, if this information wasn't printed, s/he would have to compare the paper versions with diagrams s/he sees on the screen.

7.10 Print the current context

According to cognitive integration principle, one should always be aware of the context of the current diagram. By context, we mean all diagrams belonging to the same project. As it is explained in Chapter 5, dedicated to the analysis of *Tropos/TAOM4e*, the current context is visible in the tool and, especially, in the *Outline tab*. However this part of the tool isn't visible on printed and exported diagrams.

As the problem is the same as in the case of diagrams types and names, the solution proposed is the same: put the information on exported and printed diagrams (see Table 7.17). This solution should help the reader to understand where the diagram s/he is analysing is situated in the complete network of diagrams, also if s/he consults it without using the tool, in other words, on a exported or printed version.

Table 7.17: Print current context

Title	TD10: Current context visible when diagrams are printed
Goal	Contribute towards cognitive integration of the different diagrams
Description	Print the context (surrounding diagrams, *etc.*) on exported/printed diagrams

7.11 Add a navigational map

This solution is proposed in [21]. It is linked to the cognitive integration. Indeed the goal of such a map is, as it is visible in Table 7.18, to show the complete network of diagrams and the links among them.

Table 7.18: Navigational map

Title	TD11: Navigational map
Goal	Contribute towards cognitive integration of the different diagrams
Description	Add a navigational map providing a view of the complete network of diagrams and the links among the different diagrams

The solution presented in Section 7.10 isn't completely satisfactory and could be improved with a navigational map. Indeed, it only provides the names and types of the different diagrams of the project. This information isn't generally sufficient to understand the complete network of diagrams.

The navigational map should thus show the entire network of diagrams and provide a way to navigate among them. However, we don't explain it in detail here because it was already done in Section 6.2.

Users should also have the opportunity to print this map separately. In other words, it means that modellers should have the possibility to export or print it independently of diagrams. This option could be used to avoid the overloading of diagrams but also to present the network of diagrams at the beginning of the report, if such a document exists.

7.12 Normalization

The goal of this recommendation, presented in Table 7.19, is related to the emphasis principle [21]. Indeed, the objective of the normalization is to put all elements of the same type in the same visual aspect and, consequently, avoid involuntary highlighting. According to us, the tool should thus provide a mechanism which carries out this task. This mechanism is already present in the *Preferences* window of *TAOM4e*. Indeed, in this window, there is a *checkbox* entitled Check all (apply the properties to all objects of this type). This option makes possible to set the same visual properties to all elements of the same type as the selected one. However, this mechanism has a problem: it removes the highlighting of emphasized elements.

Our solution is to add the possibility to exclude some elements from the visual normalization. This way, when a modeller wants to exclude some elements, s/he indicates it to the tool which will keep their current visual properties. Another way of proceeding is to give a list of the different constructs of the same type as the current one and ask the modeller to select the ones s/he wants to modify.

Table 7.19: Normalization

Title	TD12: Exclude some elements from normalization
Goal	Avoid involuntary highlighting
Description	Exclude highlighted elements from normalization mechanisms

The current system could also be strongly changed. We can, for example, think about a system where modellers select all the nodes they want to normalize and then access to the property window. In this case, only selected elements would be modified. There are many other alternatives like this to face this situation.

7.13 Provide a tutorial for the tool

A tutorial helps the user to learn how to use the tool, where the different options are, what are the different steps, *etc.* It is more an illustration (often based of examples) on the way the tool should be used. But, this type of help is often limited to the basic elements of a tool. The goal of a tutorial is to provide an external identification between the language and the conventions of the tool and also, as it is presented in Table 7.20, to help beginners to use the tool.

Table 7.20: Tutorial

Title	TD13: Video showing how to use the tool available offline
Goal	Address the identification principle
Description	The online video showing how to use *TAOM4e* should be available offline (for instance, for people who don't always have acces to Internet)

A tutorial for *TAOM4e* is provided online (`http://sra.itc.it/tools/taom4e/`). But it can be useful to get access to it from everywhere, also when no Internet connection is available. Our recommandation is thus to make this video available offline. An ideal option would be to include it inside the installation package of the tool. However, the video probably has a big size and it can bother users to download it. So the best solution would probably be to make possible to download this video separately.

7.14 Provide a manual for the tool

The goal of a manual is to help the user of the tool to understand it (see Table 7.21). *TAOM4e* doesn't have such a document.

Our recommendation is thus to write a manual for the tool. When one gets stuck, one generally takes the manual and looks for the useful information. If

Table 7.21: Manual

Title	TD14: User guide
Goal	Address the identification principle
Description	One should write a manual for *TAOM4e*. Such a document is a reference for novices as well as for experimented users

one owns such a document one knows where one will certainly find the correct information. It prevents one from wasting time in searching information on the Internet or somewhere else. Such a document should thus be witten for *TAOM4e*.

7.15 Allow textual comments

In *TAOM4e* it is possible to add textual comments on diagrams. It is done by the *Comment* construct. Modellers can add extra information about constructs, hypotheses, *etc.* They are also able to give shorter names to constructs and add extra information if necessary. The goal of a comment is thus to provide extra information and, consequently, act as an external identification mechanism [21]. However this mechanism of *TAOM4e* has, as it is summarized in Table 7.22, some weaknesses.

Table 7.22: Comments

Title	TD15: Link comments to diagram elements
Goal	Improve external identification
Description	If several comments are situated close from each others, one can associate those comments to a wrong construct. Our solution is thus to link the comment and the related construct

This feature of *TAOM4e* could be slightly improved. If a modeller has to put two (or more) comments for elements which are close from each others, a risk exists that the reader inverts the associations between the comments and related constructs of the diagram. The example of Figure 7.6 shows our recommendation: link the comment to the related construct, which is, in our example, the Meeting participant actor.

7.16 Provide an index of elements

As it is explained in Section 5.4, the *Outline tab* of *TAOM4e* can be used as an index. The problem is that the amount of information in this part of the tool becomes quickly huge and, by consequent, difficult to analyse. Moreover, it is impossible to know if an element is present in several diagrams. The reader has

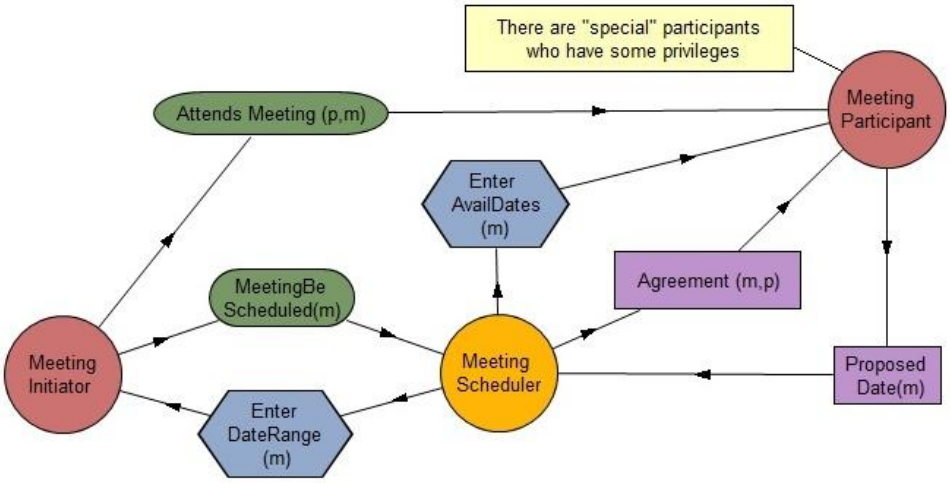


Figure 7.6: Linked comment

to have a look to all diagrams to get this information. It implies an increased rate of wasted time, mistakes, *etc.*

Another disadvantage is that this *Outline tab* isn't printable or included in printed and exported diagrams. Users don't have access to this, still imperfect, solution outside of the tool. The same problems have already been encountered previously with the names and types of diagrams, *etc.*

We propose, as it is summarized in Table 7.23, to automatically generate an index of elements present in the different diagrams to solve all those problems. The information could be structured, for example, like it is illustrated in Table 7.24. In this example there are two *Tropos* constructs, their type is indicated in the second column while the types and names of the diagrams where they appear are listed in the third column.

Table 7.23: Index

Title	TD16: Automatic index
Goal	Contribute towards integration
Description	An index helps one to know where one can find a given element in the set of diagrams One should also make a machine-readable index which could be directly imported, for example, in reports (TD16.1)

This new solution can still be improved to achieve a better usability. It could be machine-readable by using e.g. XML. This way, the index can be reused in several contexts: included in a report, included in a Web page which presents the project, *etc.*

Table 7.24: Index proposal

Name	Type	Diagrams names and types
Meeting participant	Actor	Model w/o automatic scheduler: Early req. Model with automatic scheduler: Early req. ...
Meeting scheduler	Syst. actor	Model with automatic scheduler: Early req. ...
...	...	...

7.17 Facilitate the transition between *Tropos* phases

As it is visible in Figure 7.7, the 5 *Tropos* phases are visible into the *Outline tab* of *TAOM4e*. This part of the tool indicates where the diagram one is currently making, reading, *etc.* is situated in terms of *Tropos* phases. It means that the modeller clearly knows that s/he is making, for example, an *Early requirements* diagram and s/he can easily switch to a diagram of a previous phase in order to check some properties, *etc.* Readers can also use this functionality in order to, for example, clarify some points in a current diagram by using another from a previous phase.

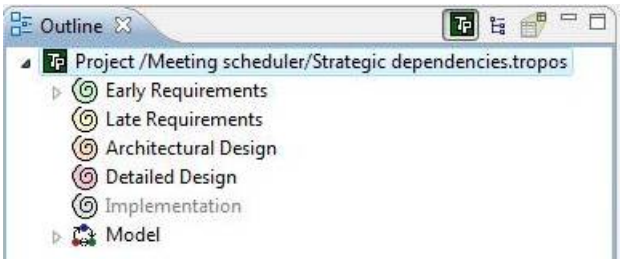


Figure 7.7: *Tropos* phases in *TAOM4e*

Our suggestion, presented in Table 7.25, is to enforce the modeller to go through all *Tropos* steps. This solution is beneficial for her/him because s/he clearly sees the logical transformation from a diagram to another, s/he probably makes less errors, *etc.* This situation is also profitable to readers: they clearly understand the transition from a diagram to another and will probably have a better understanding of the logical links between diagrams.

7.18 Make the proposed printing features optional

This last recommendation is more an advice. It is linked to the new features which print something like the index, the navigational map, the legend, *etc.* on exported and printed diagrams. It must be a choice made by the user. It's never

Table 7.25: *Tropos* steps

Title	TD17: Enforce modellers to go through all *Tropos* steps
Goal	Facilitate the perceptual integration
Description	One should go through all phases in order to have a logical continuation among the different diagrams

good to enforce her/him to do something in a completely predetermined way.

A modeller can, for example, want to print each diagram separately and have the index, navigational map, *etc.* on different sheets. This manner of working can also be useful if a report has to be made. In this case, the writer will probably present the network of diagrams first, then will structure them according to the order of the *Tropos* phases, *etc.* and finish by the index. Making this set of features optional can also facilitate the work of the report writer.

7.19 Summary

This chapter presents recommendations for tool developers. Their goal is to address the cognitive effectiveness of *Tropos* diagrams. The list presented hereunder synthesizes those recommendations.

- TD1 : Pay attention to the absolute discriminability
  - TD1.1 : Minimal size
  - TD1.2 : All elements have the same size
  - TD1.3 : Change background
  - TD1.4 : Contrast
  - TD1.5 : Minimal distance
  - TD1.6 : Automatic reorganization
- TD2 : Pay attention to the relative discriminability
  - TD2.1 : Different colours for *Actors*, *Agents*, *Roles* and *Positions*
  - TD2.2 : Check that colours are different (inter construct types)
  - TD2.3 : Check the unicity of colours (intra construct type)
- TD3: Hide & show decomposition trees
- TD4: Use different layers
- TD5: Hide some elements
- TD6: Automatic highlighting
  - TD6.1: Personalization of automatic highlighting
- TD7: Delimiting regions

- TD8: Legend on each diagram
  - TD8.1: Update legend according to used elements
  - TD8.2: Update legend *wrt* visual properties
  - TD8.3: Modifiable legend
- TD9: Print diagram name & type
- TD10: Current context visible when diagrams are printed
- TD11: Navigational map
- TD12: Exclude some elements from normalization
- TD13: Video showing how to use the tool available offline
- TD14: User guide
- TD15: Link comments to diagram elements
- TD16: Automatic index
  - TD16.1: Machine-readable index
- TD17: Enforce modellers to go through all *Tropos* steps

In the next chapter, we present recommendations for modellers who use *Tropos* and *TAOM4e* to create and maintain goal models.

Chapter 8

Recommendations for modellers

In this chapter, we present a list of recommendations which could be given to *Tropos* modellers. Modellers are people who draw diagrams. They generally have a good knowledge of the language, *Tropos* in our case, and the tool they use to build models in this language.

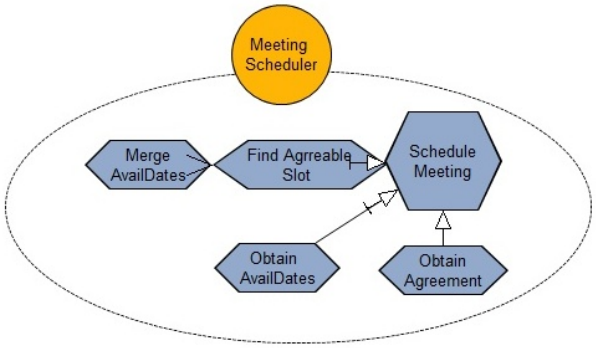
The goal of the recommendations presented here refers to the communication between the modeller and the stakeholders modelled in the project. Indeed, modellers are specialists in the goal modelling domain while their clients generally don't know anything about it. All the recommendations will be illustrated with bad examples issued from the *Meeting Scheduling System* example. Some of them are exaggerated but their goal is to show the influence of "bad" practices.

8.1 Make a good use of proximity

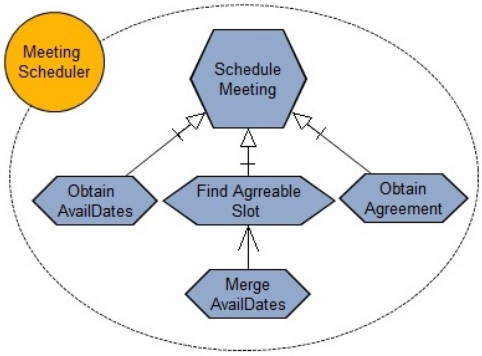
The recommendation is summarised in Table 8.1. The example in Figure 8.1(a) shows that, if elements are too close from each others, it is very difficult for the reader to distinguish e.g. *and* from *or-decompositions*. When there is a normal distance between elements, such an ambiguity doesn't occur. In Figure 8.1(a), it is also difficult to see that there is a *means-end* link between Merge AvailDates and Find Agreeable Slot. This problem is solved in Figure 8.1(b) because elements are less close from each others.

Table 8.1: Proximity

Title	MO1: Make a good use of proximity
Goal	Increase the discriminability
Description	If elements are too close from each others, they can be difficult to distinguish and one is less encline to understand the structure of the diagram



(a) Bad example



(b) Good example

Figure 8.1: Use of proximity

Those two figures also illustrate the influence of proximity on the structuration of information. The structure can be understood at the first sight in Figure 8.1(b). While Figure 8.1(a) requires more attention and is more subject to misunderstanding.

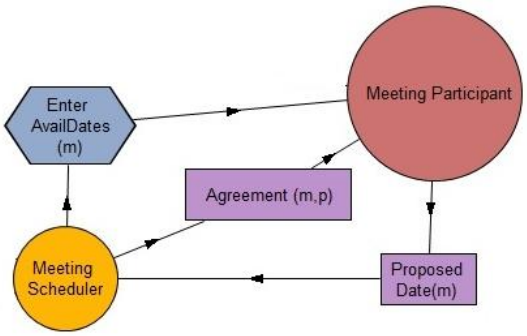
8.2 Pay attention to the size of elements

The size of the shape can be used to highlight a construct. It's due to the fact that size is one of the visual variables [21]. But modellers have to pay attention to this value (see Table 8.2). If a construct is bigger than others of the same type, the reader will tend to consider it as more important. Modellers have to take care of involuntary highlighting, especially when they use *TAOM4e* : the default size of an element is defined by the length of its label. It is modellers' responsibility to set the size of all elements of a certain type to the default size of the biggest one. In *TAOM4e*, it can be done by using the *Set default* button in the *Preferences* window.

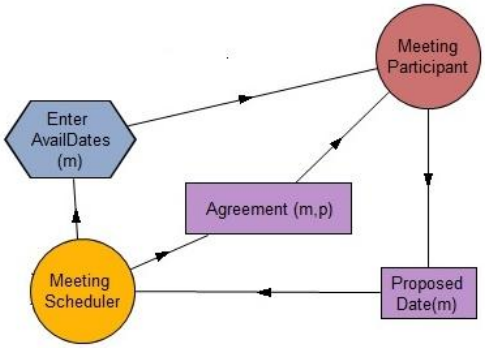
Table 8.2: Size of elements

Title	MO2: Pay attention to the size of elements
Goal	Avoid involuntary highlighting
Description	If an element is smaller than another, one can think it has less importance

Figure 8.2(a) is an illustration of bad use of the size of an element. The Meeting Participant actor seems more important *wrt* its size whereas it has the same importance as the Meeting Scheduler. For this reason, a modeller should model the situation as it is done in Figure 8.2(b). As actors have the same size, they will be interpreted as having the same importance [21].



(a) Involuntary highlighting



(b) No involuntary highlighting

Figure 8.2: Size of elements

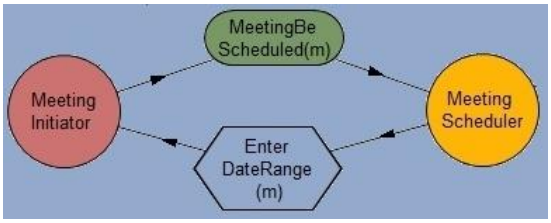
8.3 Pay attention to the contrast between elements and background

The greater the contrast between diagram elements and the background, the more readily objects will be detected and recognized [21]. Modellers have, as it is summarized in Table 8.3, to pay attention to this contrast if they want to communicate efficiently with their end-users. It implies that, if they want to respect this principle, they have to use completely different colours for the background and the constructs. For example a light coloured background with dark coloured constructs and conversely.

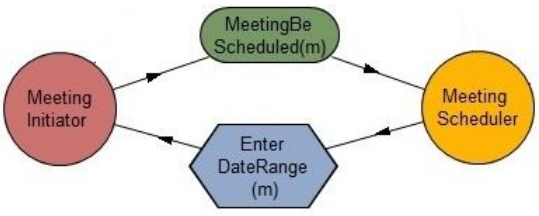
Table 8.3: Contrast

Title	MO3: Pay attention to the contrast between elements and background
Goal	Facilitate absolute discriminability
Description	If a construct and the background have the (nearly) same colour, it's difficult to see the construct

If, at is illustrated in Figure 8.3(a), an element has the same colour as the background (in this case, Enter DateRange(m)), it is difficult for the reader to see it at the first sight. It requires more concentration. For this reason, modellers should avoid choices of colours made in Figure 8.3(a) and prefer choices like the ones of Figure 8.3(b).



(a) Contrast problem



(b) No contrast problem

Figure 8.3: Contrast between background and constructs

8.4 Keep all the elements of the same type in the same visual aspect

If all constructs of a certain type have different values for their visual variables it becomes difficult to understand that elements have the same type. For this reason, as it is presented in Table 8.4, modellers have to keep the same values for the visual variables of all constructs of a certain type, excepted in case of highlighting.

Table 8.4: Visual aspect

Title	MO4: Keep all elements of the same type in the same visual aspect
Goal	Facilitate grouping of elements
Description	As the visual aspect of constructs influences their grouping by human mind, modellers should use the same visual properties for all constructs of a given type

As one can see in Figure 8.4(a), it is very difficult to see that some constructs have the same type. The single cue is the shape. While, in Figure 8.4(b), the diagram contains the same constructs but, there, they respect our recommendation and one can more easily see the different families of constructs.

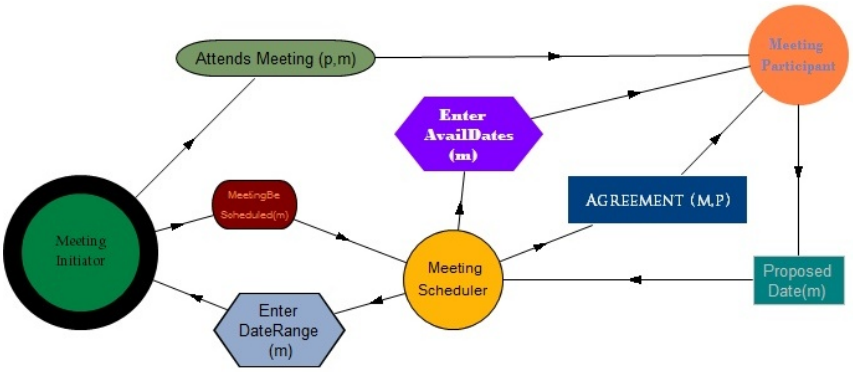
8.5 Use grouping principles

Most useful *Gestalt* laws describing how forms are perceptually organized are proximity, similarity and common region [21]. Currently, modellers only have to take care of the two first parameters because there is no way of defining regions in *Tropos* (see Table 8.5).

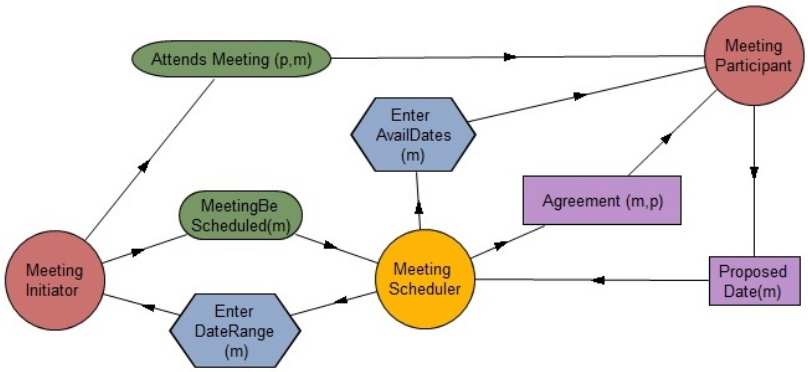
Table 8.5: Grouping principles

Title	MO5: Use grouping principles
Goal	Help the user to structure the information
Description	Modellers must pay attention to the visual properties and the position of elements because those parameters influence the structuration of the diagram by human mind

Elements close from each others will be interpreted as logically linked. This is proximity. A bad use of this parameter can lead to a bad interpretation. Evenly, elements having the same visual aspect will tend to be grouped together. It means that if, for example, an *Actor* and a *Plan* constructs have the same colour, human mind will tend to group them together. Although there is no link between them.



(a) Bad use



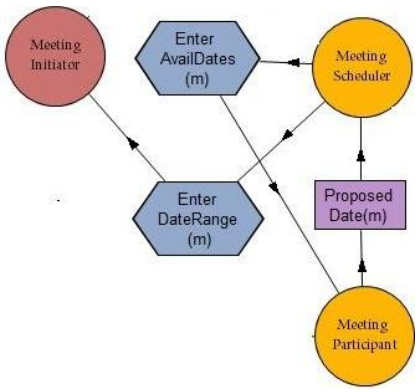
(b) Good use

Figure 8.4: Use of visual properties

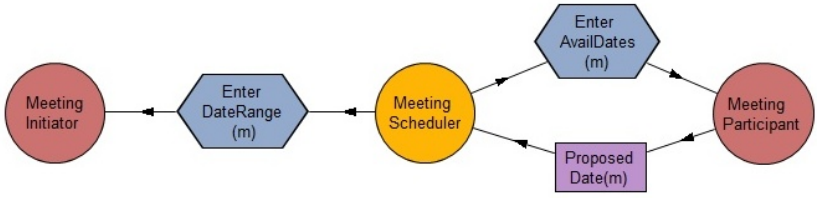
Figure 8.5(a) illustrates a bad use of proximity and similarity. The two actors Meeting participant and Meeting scheduler will probably be grouped together because of their colour while Meeting participant should be associated with Meeting initiator because they are not system actors. The second problem is illustrated by the Enter AvailDates(m) plan. The first impression of the reader will probably be that this plan links Meeting initiator and Meeting scheduler actors because of its position inside the diagram. So, with this example, one sees that one has to pay attention to the proximity and similarity in order to avoid readers' confusion. Modellers should model the diagram of Figure 8.5(a) as it is done in Figure 8.5(b) in order to reduce the error rate.

8.6 Highlight important elements

When modellers make models, some elements may be more important than others. So, the idea presented in Table 8.6 is to put the emphasis on them. One can use visual variables in order to highlight an element: a different colour, a bigger size, a thicker border, a bold font, *etc.* This difference should attract the attention of the reader. For example, in Figure 8.6, the Meeting Initiator actor



(a) Bad use



(b) Good use

Figure 8.5: Use of grouping principles

has been highlighted with a thicker red border and a blue bold label.

Table 8.6: Highlight important elements

Title	MO6: Highlight important elements
Goal	Respect emphasis principle
Description	Modellers should use visual variables like colour, border thickness, size, *etc.* in order to focus readers' attention on important elements.

Note that highlighted elements can be different from a reader to another. In the *Meeting Scheduling System* example, important elements depend on the interlocutor. If one discusses with a Meeting initiator, one doesn't care about the exchanges between the Meeting scheduler and the Meeting participant. On the other hand, a Meeting participant doesn't care about the relationships between the Meeting scheduler and the Meeting initiator. Modellers should adapt their models to the situation.

Figure 8.6: Highlighting example

8.7 Use comments in order to improve comprehension of the user

Modellers also have the opportunity to add comments in *Tropos*. They can thus add extra information which can be considered as a part of the documentation of the model. In *TAOM4e*, those notes are represented by yellow rectangles. This recommendation is summarized in Table 8.7 and was already illustrated in Figure 7.6.

Table 8.7: Comments

Title	MO7: Use comments in order to improve comprehension of the user
Goal	Improve the external identification
Description	Modellers should use comments if they can't express something in a graphical way. But they have to avoid superfluous information

Modellers have to pay attention to not include superfluous information in their comments. This will disturb the reader and will complicate her/his understanding of the model. Moreover, it exists a hazard that s/he focuses on the comments while s/he doesn't care about more important elements. So, modellers can use comments but must do it carefully.

8.8 Document your model

As it is impossible to express everything in a graphical way, a documentation of the model is still required. Such a document should, as it explained in Table 8.8, always come with a model. It should contain more information about the model itself.

Such a document should, for example, include a complete description of each construct: *Actors*, *Hardgoals*, *Plans*, *etc.* This description is required to agree on the definition of an element. For example, in Figure 8.4(b), there is a *Plan* construct labelled Enter AvailDates(m). One can wonder what this title means. One probably understands that the task is to provide availability dates. But one probably doesn't know what the m parameter represents: a date, a period,

Table 8.8: Model documentation

Title	MO8: Document your model
Goal	Improve the external identification
Description	Elements presented in a diagram usually have short names conveying only a part of the information. The information which can't be expressed in a diagram should be written into the documentation of the model

etc. Such an information should be clearly explained in the documentation of the model.

However, diagrams still remain the object used for the discussion between the modeller and her/his interlocutors. The documentation is present as a support if something isn't clear in the diagram.

8.9 Learn language and tool principles from documentation

The eight previous recommendations aim at improving the quality of models made by *Tropos* modellers. But they aren't useful if the language and the tools aren't used according to their manuals, semantics, *etc.*

Tropos has a syntax and a semantic which must be respected [6]. It also has a meta-model [11]. Consequently, modellers aren't free to do whatever they want. The first step is, as it is presented in Table 8.9, to learn the language to see what it is possible to do with it.

Table 8.9: Learn tool & language

Title	MO9: Learn language and tool principles from documentation
Goal	Improve external and internal identifications
Description	One should always learn the *Tropos* syntax, semantic and meta-model. And, if one uses *TAOM4e*, one should also learn how to use it

The usage of documentation is an help for the modeller. It isn't required but it can prevent her/him from wasting time. Tools like *TAOM4e* can improve the quality of models: they are made in such a way that they constrain the possible actions of modellers according to the principles defined in the meta-model of *Tropos* [11]. However, modellers must still keep in mind the principles of *Tropos* because tools can be imperfect.

8.10 Summary

In this chapter, we proposed recommendations for modellers based on principles given in [21]. As modellers communicate directly with the stakeholders of the modelled domain, their models have to respect principles for effective diagrams. The list hereunder summarizes those recommendations.

- MO1: Make a good use of proximity
- MO2: Pay attention to the size of elements
- MO3: Pay attention to the contrast between elements and background
- MO4: Keep all the elements of the same type in the same visual aspect
- MO5: Use grouping principles
- MO6: Highlight important elements
- MO7: Use comments in order to improve comprehension of the user
- MO8: Document your model
- MO9: Learn language and tool principles from documentation

This chapter finishes the second part of the thesis. Here we presented our analysis of *Tropos / TAOM4e wrt* effective communication principles and we gave some recommendations for improving the language and the tool.

In the next part, we validate the different recommendations.

Part III

Validation

Chapter 9

Illustrative example: *Conference Management System*

In this chapter we illustrate the application result of the recommendations. It is be done with an example called *Conference Management System*. The goal, here, is to illustrate the proposals made by applying the recommendations for *Tropos* language engineers, tool developers and modellers on the *Conference Management System* example.

The *Conference Management System* example is presented in Section 9.1. It presents the main principles of the domain, the actors, their relationships, *etc.* This part aims at defining all the concepts in order to get users' understanding. Then, in Section 9.2, the different diagrams of the *Late requirements* phase as discussed in [22] are presented. Further in this section, we apply the recommendations presented in Chapters 6, 7 & 8 to those diagrams and show the limitations of our example. At the end of the chapter, in Section 9.3, we compare original diagrams and modified ones. By modified diagrams, we mean diagrams which are modified according to our recommendations.

9.1 The *Conference Management System* example

Before starting to illustrate how we apply our recommendations of Chapters 6, 7 & 8, we have to introduce the *Conference Management System* example which is, as it is clear from its name, a system which helps to manage conferences. We need to understand what a conference is, who are the different stakeholders involved in the complete process, what are their interactions, *etc.* All those questions are clarified in this section. A complete understanding of the domain is required in order to completely understand the different diagrams of Section 9.2. The definitions presented below are adapted from [2, 12, 28].

9.1.1 What is a *Conference*?

A *conference* is a meeting which generally welcomes more than 250 participants and runs several days. A meeting, in this context, is a scientific event assembling researchers [28]. However some conferences can have less or more participants and can also last less or more days.

The set of topics of the *conference* can be horizontally or vertically specialized. In the vertical division, the set of topics is oriented on a very specific domain, application, *etc.* While the horizontal decomposition allows a broad range sets of topics.

The general process of a conference can be divided as follows. First, the organizers have to find reviewers for the submitted papers. Secondly, they have to make a call for paper in order to get papers for their conference. The papers are then submitted by different authors. The organizer collects them and reviewers review them. Authors are then contacted in order to be informed whether their paper is accepted or not. If a paper is accepted, its author(s) has/have to prepare a camera-ready version for the conference. Then, *Proceedings*, which are the collection of the significant papers presented at a meeting or a summary of all the papers presented during the meeting, are transmitted to the publisher.
Organizers also have to manage logistic aspects of the conference like the place, the schedule, the accommodations, *etc.*

A *conference* may include *symposia*, *workshops* and/or *tutorials* but these concepts aren't explained here. It isn't useful for our example. They are just other types of meetings.

9.1.2 Who are the different stakeholders?

The goal of this section is to describe the main actors of the *Conference Management System* example which will be used in the following sections. Readers need to get a complete comprehension of those actors in order to understand the different diagrams presented in Section 9.2. There are 4 main different actors.

The first one is the *Author*. It is someone, often a researcher, who participates in the writing of a paper. It means that s/he writes the document, supervises the researches, rereads the final version in order to correct the spelling, *etc.* In short, a person who takes part in the process ending in the delivered paper. A synonym of *author* can obviously be *co-author*.

The *Program Committee* (or PC) is, as it is indicated in its name, in charge of the program of the conference. It means that it has to determine the topic of the conference and arrange its technical program. It makes it in collaboration with the sponsors (if any) and the *Conference Committee* which is responsible for the global organization of the meeting.

Another actor is the *Reviewer*. S/he is a researcher, usually expert in the domain of the submitted paper, who evaluates its scientific quality and reports

to the *Program Committee*. Generally several *Reviewers*, also called *Referees*, are associated with a single *conference*.

Finally, the last actor presented here is the *Publisher*. Her/his role is to get the proceedings in order to publish them for the *Program Committee*.

We also have to mention that, according to [2, 12, 28], there are other actors involved in the global process of a conference. So, for example, committees like *Publicity and Public Relations Committee, Finance Committee, etc.*, roles like *Conference Chair, Treasurer, Program Chair, etc.*, weren't explained in this section. The point is that the example of Section 9.2 involves only a part of the set of actors presented in [2, 12, 28]. So, we only described actors represented in this example in order to avoid users' confusion.

9.1.3 What are the interactions among the different stakeholders?

The best way to get users' understanding of the relationships among the 4 actors of the conference management process is probably to explain it with an example. Figure 9.1 represents the UML activity diagram of the paper selection process [28]. This diagram isn't complete *wrt* the complete process of a conference because it represents only the elements used in our analysis of Section 9.2. So, for example, the logistic of the conference like the place, the schedule, the accommodations, *etc.*, aren't presented here.

The 3 different columns, called *swimlanes* in UML Activity diagrams, represent 3 different actors. The first column contains the *Author*, the second the *Program Committee* and the last one, the *Reviewer*. The *«multiple»* value, called *stereotype* in UML Activity diagrams, for the *Author* and the *Reviewer* means that there can be several actors of this type included in the process, all of them making the same activities as the one represented in their swimlane. An attentive reader will probably notice that an actor presented above is missing: the *Publisher*. It isn't included in the diagram but her/his role is, as it was explained in Section 9.1.2, to get the proceedings and publish them for the conference.

One can also notice that this UML Activity diagram has been divided in 3 subdiagrams, *A, B & C*, by dotted rectangles. Those subdiagrams respectively represent the call for paper and seek of reviewer (A), the review itself (B) and, finally, the final steps of the process (C).

In the first subdiagram (A), the *Program Committee* starts the process and it carries out two activities: Publish call for papers and Invite potential reviewer.

The first activity produces a Call for paper object which can be an e-mail, an advertisement in a scientific journal, *etc.* Its goal is simply to inform interested people that there is such a call. When concerned people get this information, their action is to Submit a paper. This activity often includes the writing of the Paper which is then sent to the *Program Committee*. There can be several papers (*«multiple»* stereotype) and they are, at this moment, all considered

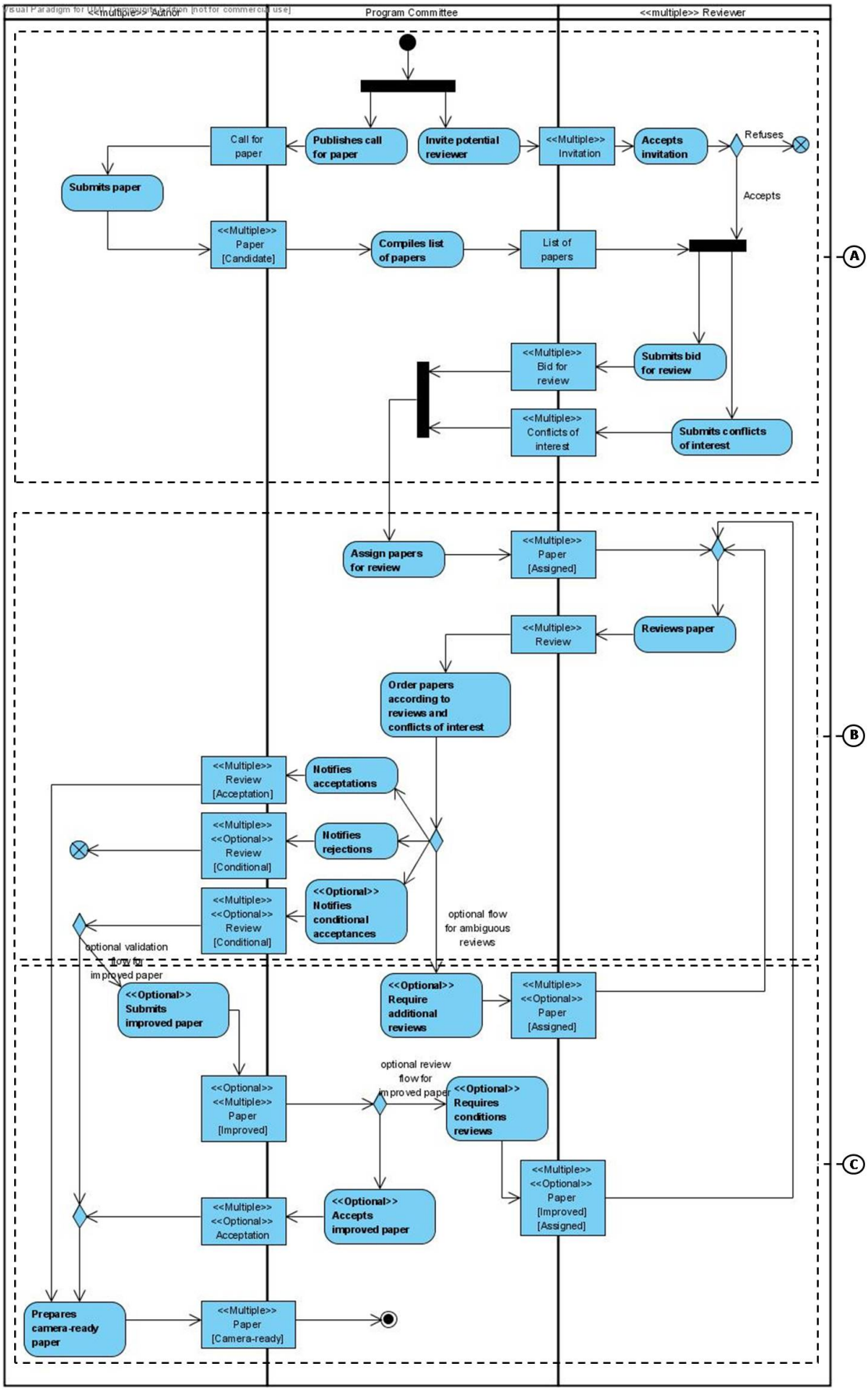


Figure 9.1: Activity diagram of the paper selection process

Candidates because they still have to be approved.

On the other side, the Invite potential reviewer activity produces multiple Invitations for reviewers. It means that the *Program Committee* has to get in touch with *Reviewers*. Those lasts have then to decide if they accept (or not) the invitation. It is illustrated with a diamond. If the answer is negative, it means that the Refuses output of the diamond is chosen, then the flow is ended. On the other hand, if the reviewer Accepts, s/he has to wait for the List of papers which is supplied by the *Program Committee* which Compiles the list of papers provided by the different *Authors*.

When *Reviewers* get the papers, they carry out two activities: Submits bid for review and Submits conflicts of interest. Both results, Bid for review and Conflicts of interest, are sent to the *Program Committee*. The merger of both objects ends the first subdiagram (A).

The second subdiagram, labelled B, represents the review process itself. The committee Assigns papers for review to the different *Reviewers*. The Paper object is thus transmitted to the *Reviewer* who carries out the Reviews paper activity.

The *Program Committee* receives the Review objects made by the different *Reviewers*. They have then to carry out the Order papers according to reviews and conflicts of interest activity. At this point, there are 4 possibilities. The first, and simplest one, is to accept the given submitted paper and, in this case, the *Program Committee* Notifies acceptation to the author(s). Another is to Notify rejection of the paper to the writers. In this case, the flow is ended. The third possibility aims at clarifying some reviews which could be ambiguous. In this case, the paper Requires additional reviews and the Paper object is transmitted to *Reviewers*. This possibility isn't mandatory. The last possibility is situated between the two firsts: The *Program Committee* Notifies conditional acceptance and a Conditional acceptance object is transmitted to the authors.

Finally in the last subdiagram (C), when an *Author* receives a conditional acceptation, s/he has the opportunity to Submit improved paper. The *Program Committee* receives the Paper object and has then two options for the improved papers. Either it Accepts improved paper itself. Either it Requires conditions reviews and the Paper object is sent to the reviewers. In that case, the process of subdiagram B is restarted.

Finally, when a paper of an *Author* is accepted (directly, improved or conditionally), s/he Prepares a camera-ready paper and transmits this Paper object to the *Program Committee*. This camera-ready version will be used during the *Conference*. It's the final activity node.

9.2 Application

As it has already been mentioned in the introduction of this chapter, we illustrate the modifications generated by the recommendations of Chapters 6, 7 & 8. The domain of this model is the management of a conference as it was presented in the previous section.

This part is based on a *Tropos* model of the *Conference Management System* as discussed in [22]. The next subsection includes the *Late requirements* diagrams of this model as they where drawn by modellers. Their modified versions made according to the recommendations explained in Chapters 6, 7 & 8 are presented in Section 9.2.2. The fact that the original version of the different diagrams is used means that this example wasn't made on purpose. The objective of this confrontation is to show the benefits of the proposed changes. This way of presenting diagrams enables the comparison.

The suite of the section will be divided in 3 parts: one for the current versions of the diagrams, a second one for the modified diagrams and a last one presenting some recommendations which can't be illustrated in the selected example.

9.2.1 *Current version of diagrams*

The current version of *Late requirements* diagrams is presented in Figures 9.2 and 9.3 which come from [22]. Figure 9.2 represents the *Actor model* while the second shows the *Goal model* of the *CMS System*.

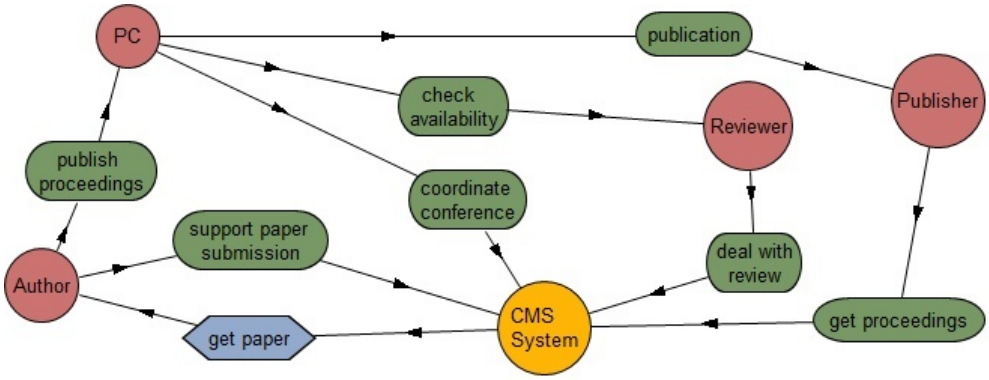


Figure 9.2: Original actor model

According to principles presented in [21], diagrams of Figures 9.2 & 9.3 have some weaknesses. They are summarized in Table 9.1.

The principle concerning discriminability is nearly respected. The absolute discriminability is evaluated by 3 variables: size, contrast and proximity.
PC, Author, accept and reject have small sizes, according to the criteria given in [21], which is determined by their label. The size of other elements is sufficient according to the same principle.

Table 9.1: Evaluation of current diagrams

Principle	**Factors**	**Evaluation**	**Conclusion**
Principle #1	Size	OK excepted PC actor, accept and reject plans	Size of elements and proximity could be improved
	Contrast	White background and coloured elements	
	Proximity	Some elements of Figure 9.3 are to close from each others	
	Relative discriminability	Different types of elements have different visual properties	
Principle #2	Seven plus or minus two rule	There are 13 constructs in Figure 9.2 & 17 in Figure 9.3	There are too much elements
Principle #3	Emphasis	PC & Author are downplayed because of their size	There is involuntary downplaying
Principle #4	Perceptual integration	No navigation cue	Difficult navigation & no global view of all diagrams
	Conceptual integration	No integration mechanism	
Principle #5	Iconic representation	No icon	Tree layout improves direct understanding
	Perceptually direct relationships	Tree structure in Figure 9.3	
Principle #6	Proximity	Bad proximity of some elements	Misunderstanding possible due to the position of elements
	Similarity	Elements of the same type are similar excepted size	
	Common region	No way of delimiting regions	
Principle #7	External identification	Title & type visible only in the tool	Some elements already present but could be improved
	Internal identification	No legend	
Principle #8	Visual variables	Shape, colour, size, X&Y positions	4 visual variables used
Principle #9	Graphic simplicity	5 different constructs in diagrams of Figures 9.2 & 9.3	Under the 6 limit given by [21]

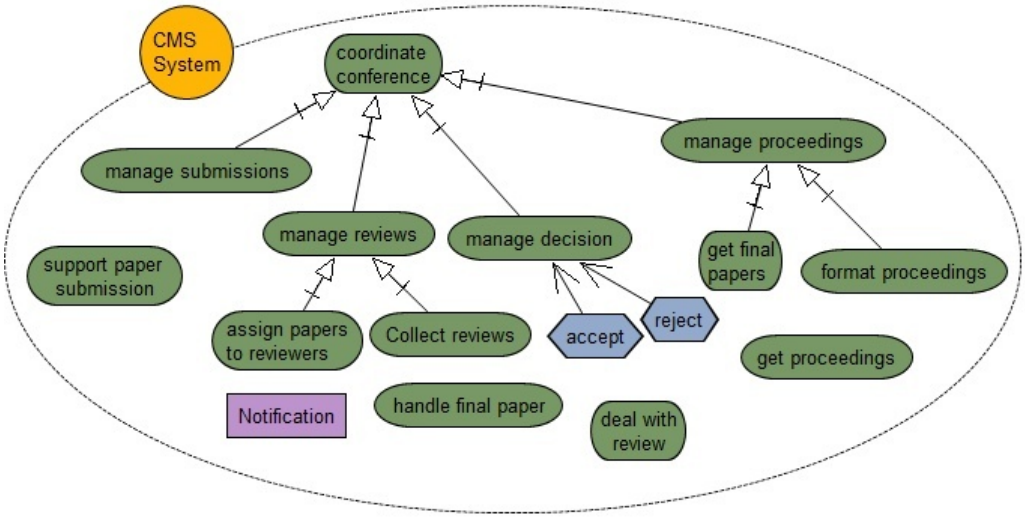


Figure 9.3: *CMS System* original goal model

There is no problem of contrast because the background is white and colours used for the different constructs are dark ones.

Some elements of Figure 9.3 like accept, refuse, get final papers are too close from each others. It is not very confusing but it can disturb some readers who are not used to the *Tropos* notation.

The relative discriminability was already discussed in Chapter 5. However, here, it is impossible to illustrate the conflict of visual properties among the different *System* constructs because only a single one is represented in the *Conference Management System* example.

Another principle refers to the number of constructs in a diagram. The diagram of Figure 9.2 contains 13 constructs while 17 are present in the one of Figure 9.3. It's high above the *7 plus or minus 2* rule quoted in [21].

There is a principle which refers to the highlighting of elements. In Figures 9.2 and 9.3, there is no highlighting. However the CMS System actor could be highlighted because it is the most important element of the *Tropos Late requirements* phase. Moreover, in Figure 9.2, one can consider that PC and Author are less important than other actors because of their size. As it has already been explained in Chapter 5, it is related to the fact that, in *TAOM4e*, the default size of a construct is determined by the size of its label. So, here, PC and Author labels are smaller than other ones. One may thus consider that PC and Author actors are less important than the 3 others which have the same size. The same problem occurs for *Hardgoals*.

The next principle is related to the cognitive integration. In the *Conference Management System* example, one doesn't know the link between diagrams of Figures 9.2 and 9.3. Obviously, if one knows the semantic of *Tropos*, one can link

both diagrams together because the first is the *Actor* model while the second the *Goal* model.

Another principle is dedicated to the perceptual directness. In the diagram of Figure 9.3, elements are positioned in such a way that one understands that Coordinate conference is decomposed in four sub-*Hardgoals* which are, in turn, decomposed, *etc.* Moreover, the symbols on the connection links indicate their orientation.

As it has already been mentioned in Chapter 5, there is no icon and stakeholders must learn that, for example, a yellow circle represents a *system actor*, a pink circle an *actor*, *etc.*

Concerning the structure principle, some elements like coordinate conference and check availability or deal with review and get proceedings of Figure 9.2 are too close from each others. The same problem occurs in Figure 9.3: the collect reviews *Hardgoal* is positioned under the manage decision *Hardgoal*. Consequently, an inattentive reader will, at the first sight, interpret collect reviews as a subgoal of manage decision while it is a subgoal of manage reviews.

In both diagrams, elements of the same type (*Hardgoals*, *Actors*, *etc.*) have the same visual aspect excepted, as it is explained in the paragraph concerning the emphasis principle, their size. This way one can easily associate elements of the same type together.

As it has already been explained in Chapter 5, there is, in *TAOM4e*, no way to delimit regions. So, modellers weren't able to use it for the diagrams presented in Figures 9.2 and 9.3.

The external identification depends on the title and the type of a diagram. Here, one can know the name of diagrams with the legend of figures but this value isn't included on diagrams when they are exported from *TAOM4e*. The same problem occurs with the type of diagrams: here, one knows that both diagrams are *Late requirements* ones because we mentioned it, but it isn't written on exported diagrams. Currently, both values depend on document writers who have to add type and name as a legend to diagrams.

There is also no internal identification. Indeed, there is no mechanism like a legend, a key, *etc.* illustrating the graphical conventions used.

In *Tropos*, 4 visual variables are used: shape, colour, size and X&Y positions. One can see that, in our example, those variables are sufficient in order to distinguish the different types of constructs.

The last principle refers to graphic simplicity. In our case, there are 5 different types of constructs in diagrams of Figures 9.2 and 9.3. This is not a large number and, consequently, doesn't require more concentration for readers according to [21].

9.2.2 *Modified versions of diagrams*

In the previous section we presented *Late requirements* diagrams of the *Conference Management System* as they were drawn by their modeller. This section

aims at presenting the same phase of the *Tropos* methodology with some modifications made according to recommendations presented in Chapters 6, 7 & 8. The result is visible in Figures 9.4, 9.5 and 9.6.

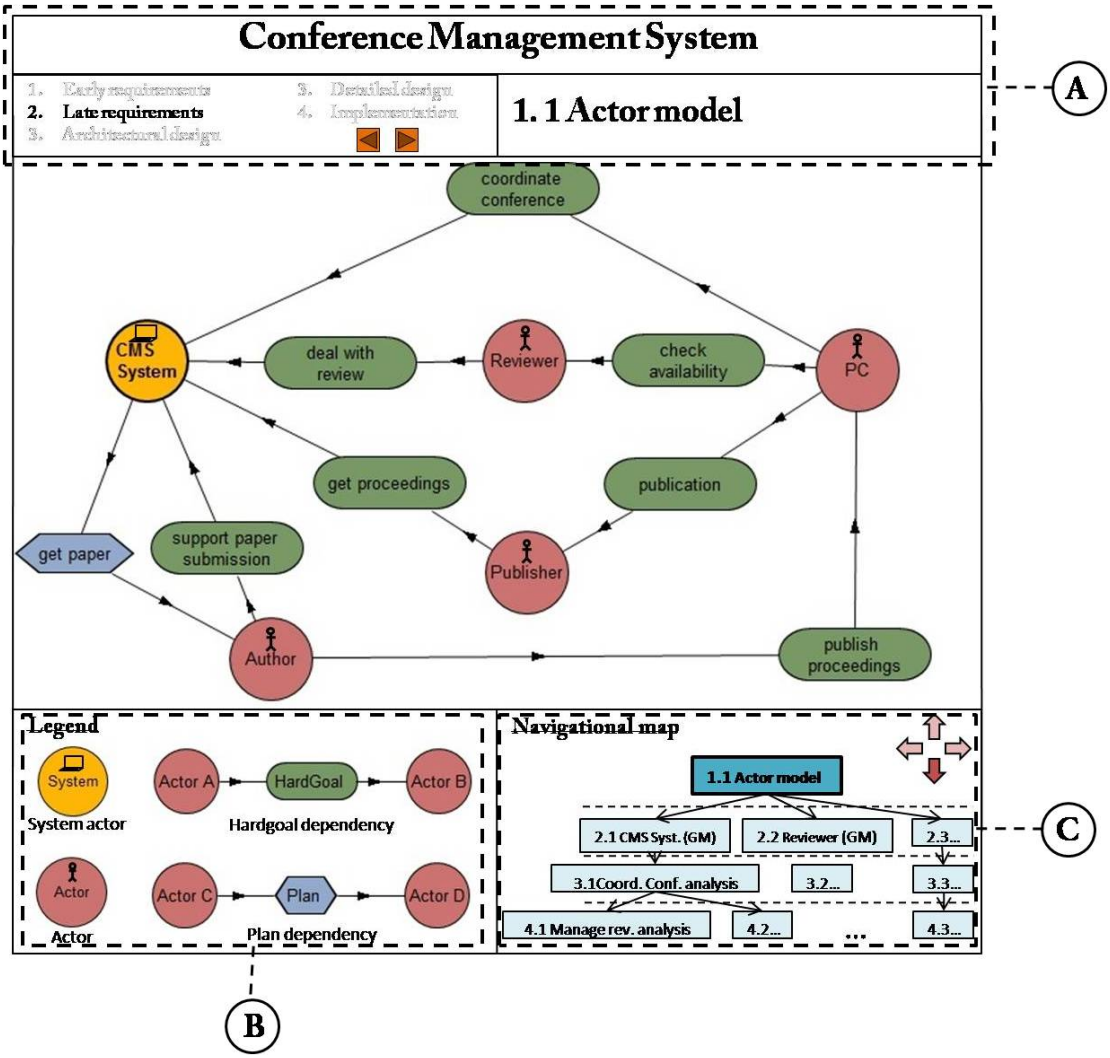


Figure 9.4: Modified actor model

Table 9.2, synthesizes the different recommendations applied to the three new diagrams. Note that the value of the third column represents the identifier of the recommendation(s) applied. A complete list of those values can be found in Section 7.19.

The discriminability principle is, in our 3 diagrams, linked to 9 different recommendations.

The size of elements was changed according to recommendations TD1.1 &

Table 9.2: Evaluation of modified diagrams

Principle	**Factors**	**Recommendation(s)**	**Comments**
Principle #1	Size	TD1.1, MO2	All elements have a minimal size
	Contrast	TD1.4, MO3	White background and dark coloured constructs
	Proximity	TD1.5	Minimal distance between elements
	Relative discriminability	TD1.2, TD2.2, TD2.3, MO4	Colour & size are discriminating variables
Principle #2	Seven plus or minus two rule	TD3	Decomposition tree hidden
Principle #3	Emphasis	TD2.3, MO6	CMS System is highlighted
Principle #4	Perceptual integration	LE2, LE3, TD10, TD11	Navigational map and cues added
	Conceptual integration		
Principle #5	Iconic representation	LE4	Two icons have been introduced
	Perceptually direct relationships	TD3	Figure 9.6 has a tree layout
Principle #6	Proximity	MO1, MO5	Risk of bad associations reduced
	Similarity	TD1.2, TD2.3, MO4	Elements of the same type have the same visual properties
	Common region	/	/
Principle #7	External identification	TD9	Names visible on exported diagrams
	Internal identification	TD8, TD8.1	Legend added
Principle #8	Visual variables	TD2.3	Icons are a new visual variable
Principle #9	Graphic simplicity	/	/

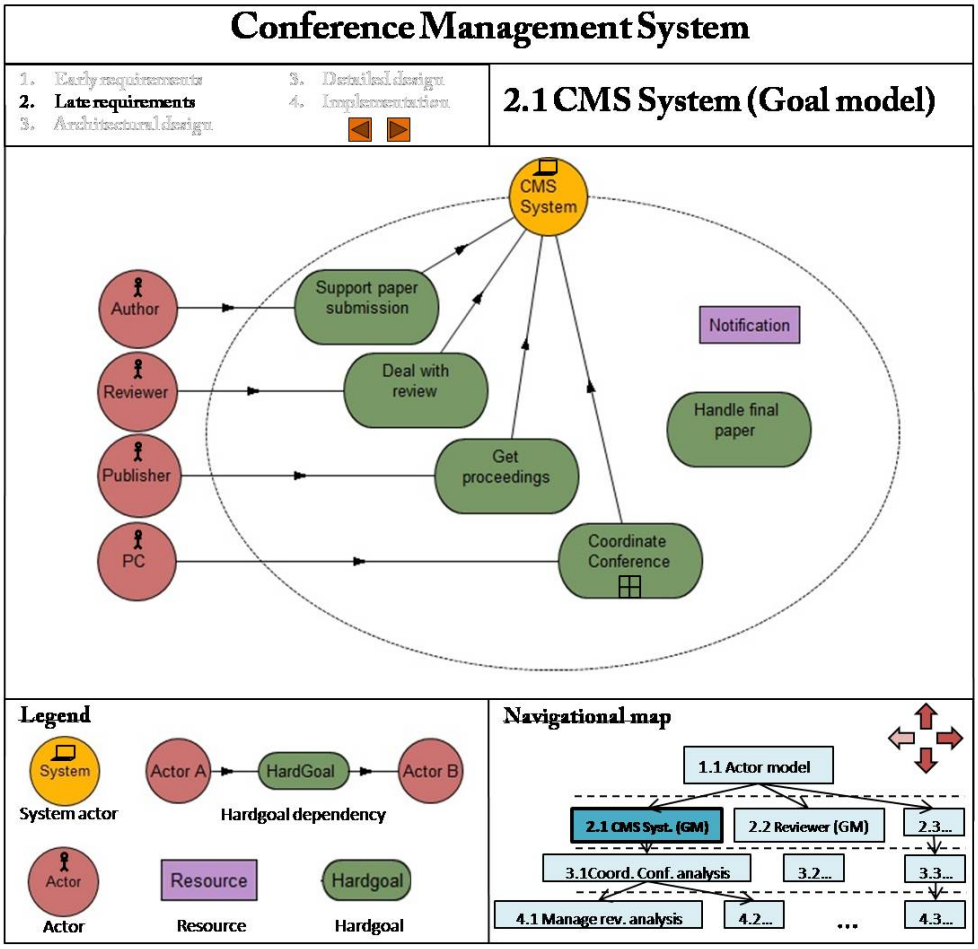


Figure 9.5: *CMS system* modified goal model

MO2. It means that, for example, Author and PC are no more smaller than other actor constructs.

Nothing has changed about the contrast because there was no problem in the original version. It means that modellers respected recommendation MO3. If this recommendation wasn't respected, the mechanism provided by recommendation TD1.4 would signal (and eventually correct) that there is a problem of contrast.

The proximity has been modified according to recommendation TD1.5. There is less risk of confusion in diagrams of Figures 9.4, 9.5 and 9.6 because constructs are sufficiently spaced to avoid misunderstanding.

Finally, the relative discriminability is related to 4 different recommendations. One can verify that, on each diagram, all constructs of the same type have, according to recommendation TD1.2, the same size. Moreover, with the mechanism provided by recommendation TD2.2, one is sure that different types of constructs have different colours. In our case, *Actor*, *Plan*, *Hardgoal* and *Re-*

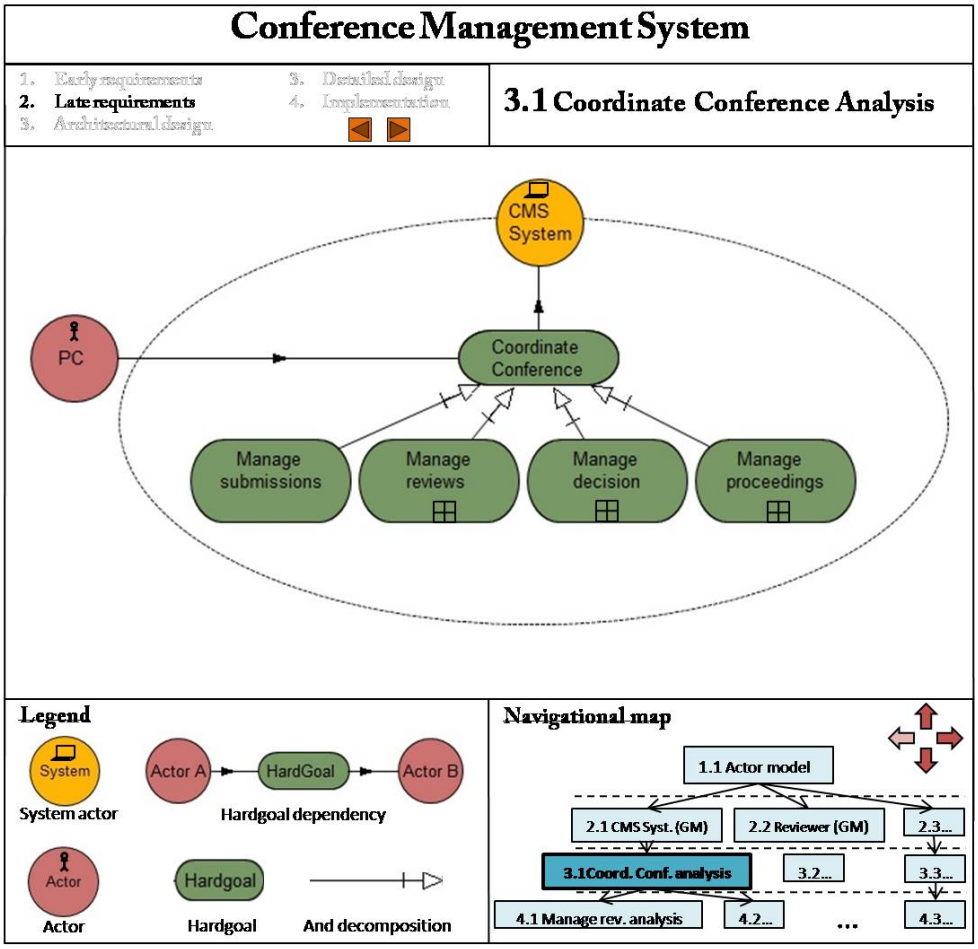


Figure 9.6: *Coordinate conference* diagram

source constructs have different colours. This property, in combination with the fact that all elements of the same type have the same colour (Recommendation TD2.3), makes possible to consider the colour as a real discriminating variable. The last recommendation, MO4, linked to the relative discriminability, also permits one to use all visual variables as discriminating ones.

Recommendation TD3, related to the modularity principle, is visible in Figures 9.5 and 9.6. Indeed, a new + symbol has been added to Coordinate conference *Hardgoal* of Figure 9.5 and to Manage reviews, Manage decision and Manage proceedings *Hardgoals* of Figure 9.6. It indicates that the *Hardgoal* will be developed in a lower level diagram. In our case, Coordinate Conference *Hardgoal* of Figure 9.5 is developed in Figure 9.6. Then, the 3 *Hardgoals* of Figure 9.6 with a + symbol, are developed in lower level diagrams (which aren't represented here). Note that the + sign is no more present on the Coordinate Conference *Hardgoal* of Figure 9.6. It's due to the fact that its decomposition

tree is developed there.

The emphasis is, in our example, linked to two recommendations: TD2.3 and MO6. In Figure 9.4, the CMS System actor is highlighted with a thicker border because it is the most important element of the *Late requirements* phase.

Two mechanisms linked to the cognitive integration are introduced. The first one is a navigational map as it was recommended in LE2 and TD11. It is situated in the bottom right corner of Figures 9.4 (Part C), 9.5 and 9.6. The navigational map of Figure 9.7 is the one of the *Actor model* (Figure 9.4). There, the box representing the current diagram (1.1 Actor model) is highlighted with a ticker border, a different colour and a bold font. This way, one knows where one is, and consequently where one can go, in the navigational map. One can, for example, go to 2.1 CMS Syst. (GM). In this case, one is redirected to Figure 9.5. Note that, in the navigational map of this diagram (Figure 9.8), the 2.1 CMS Syst.(GM) is highlighted. From this second diagram one can still access other ones like, for example, the one of Figure 9.6. As one can note on Figures 9.4, 9.5 and 9.6, this navigational map is, according to principle TD10, still visible in printed versions of diagrams.

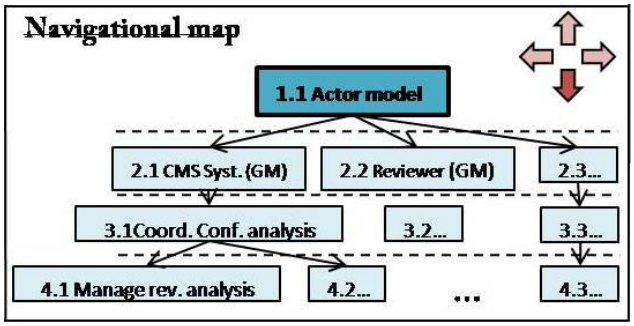


Figure 9.7: Navigational map of the actor model

Other elements were added according to LE3: navigation cues. A first one is situated in the same box as the navigational map. There are 4 arrows, enabling one to go *up*, *down*, *left* & *right* in the navigational map. Note that in Figure 9.7, only the *down* arrow is activated. It is represented by the fact that the 3 others arrows are light coloured while the *down* one is dark coloured. If ones clicks on this *down* arrow, one will be redirected to the 2.1 CMS Syst.(GM) diagram which is the first left son of the current element in the navigational map. The same navigation cue for the diagram 2.1 CMS Syst.(GM) is visible in Figure 9.8. There, the *up*, *down* & *right* arrows are activated because this diagram has a father (1.1 Actor model), a son (3.1 Coord. Conf. analysis) and a right brother (2.2 Reviewer (GM)).

A second navigation cue has been introduced in the part of the diagram containing its type. It is situated in the bottom left box of Part A of Figure 9.4. It is constituted of two arrows. They offer the possibility to go backward and forward in the different *Tropos* steps. If one clicks on the *left* arrow of Figure

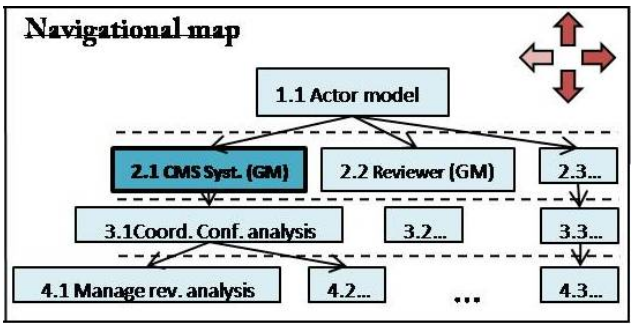


Figure 9.8: Navigational map of the CMS System goal model

9.4, one will be redirected to *Early requirements* phase diagrams. While, if one clicks on the *right* arrow, one will be redirected to the *Architectural design* phase diagrams. Once again, if one is visualizing a diagram of the first *Tropos* phase, the *left* arrow is deactivated (see Figure 9.9(a)) and, if one is at the last phase, it is the *right* arrow which is deactivated (see Figure 9.9(c)).

(a) *Early requirements* phase

(b) *Late requirements*, *Architectural design* & *detailed Design* phases

(c) *Implementation* phase

Figure 9.9: Navigation cue for *Tropos* phases

One can also notice that 2 icons have been added according to recommendation LE4. The first one was already presented in Chapter 6: a little man. It is a new variable which means that the construct is an actor. The second icon is in the same line: an iconic computer which means *System actor*.

The recommendation TD3 which was presented for the modularity principle is also linked to the perceptual directness one. Indeed, a tree representation is a way often used to represent a decomposition.

The structure principle is related to the grouping of elements. Here we placed elements according to recommendations MO1 and MO5. It means that logically linked elements are placed close from each others. In Figure 9.4, for example, deal with review *Hardgoal* has been placed just between CMS System and Reviewer actors. This way one knows, according to elements presented

in [21], that they are logically linked.

The second artifact helping to understand the structure presented in [21] is the similarity. In our case, it is represented by recommendations TD1.2, TD2.3 and MO4. TD1.2 advises to keep the same size for all the elements of the same type. One can check that in Figures 9.4, 9.5 and 9.6, all elements of the same type have the same size. TD2.3 is based on the same principle: it is a mechanism which checks that all elements of the same type have the same colour. But, according to recommendation MO4, modellers also have to pay attention to the visual properties. It means that those properties are, according to our recommendations, checked by two stakeholders: tool developers , and consequently their tools, and modellers.

The identification principle is divided in two parts in [21]: external and internal identification.

In our case, external identification is represented by two elements: diagram name and type. In Figure 9.4, this information is visible in Part A. According to recommendation TD9, the name of the project, Conference Management System, and the name of the current diagram, 1.1 Actor model, are present in the exported diagram. The type of the diagram, 2. Late requirements, is, according to the same recommendation, also visible in the same part of Figure 9.4. Moreover, one knows, with the number preceding the phase name, its position in the complete set of *Tropos* phases. Now one directly knows the name of the project, the name of the current diagram and its type.

The name of diagrams were changed by introducing 2 numbers. The first one indicates its level in a potential decomposition tree while the second one indicates its position in this level. So, for example, Figure 9.6, the name of the diagram is 2.1 CMS System(Goal model). It means, as it is visible in the navigational map, that it is at the second level and that it is the first one of this level.

In order to apply recommendation TD8, a legend was added. It is situated in the bottom left of Figures 9.4 (Part B), 9.5 and 9.6. Moreover, this legend was made according to recommendation TD8.1. Indeed, it contains only elements which are present in the diagram. For example, the legend of Figure 9.4 doesn't contain the *Resource* construct while the one of Figure 9.5 do because this element is present in the diagram 2.1 CMS System(Goal model).

Finally, we can consider that recommendation TD2.3 presented for the perceptual directness principle is also valid for the visual expressiveness. Indeed, icons are a new visual variable for *Actors* and *System actors*.

9.2.3 Limitations

As it was mentioned at the beginning of this section, we used the *Conference Management System* example as discussed in [22] in order to illustrate recommendations proposed in Chapters 6, 7 & 8. The fact that this example wasn't made on purpose implies that some proposed recommendations can't be illustrated.

Concerning language engineers three recommendations aren't illustrated. The first one is recommendation LE1 which is related to the differentiation

of diagram elements using visual variables and particularly to *Actor*, *Agent*, *Role* and *Position* constructs. The problem of these different constructs isn't visible in the *Conference Management System* example because a single type of actor is represented there. Consequently, it is impossible to show the improvements made by this recommendation. Note that this remark is also valid for recommendation TD2.1 for tool developers.

The second one is LE5 which recommends to introduce techniques to group elements like dotted squares, *etc.* This recommendation isn't illustrated here because we had no opportunity to do so.

The last one is the idea to make manuals for modellers and readers presented in LE6 and LE6.1. Obviously, to show the benefits of such a document, the best solution would be to write it completely. Moreover, an empirical study could be done. Its principle would be to divide a group of people in two subgroups. Both would do the same "exercise" excepted that the first one would read the manual while the second wouldn't. At the end, we would check the results of both groups and analyse the differences to see the influence of such a document.

It is also difficult to illustrate some recommendations made to tool developers. The main reason is that most of those proposals are dynamic ones. By dynamic, we mean that there is an algorithm behind and, to see how it works, one needs to execute it. One can only see the result.

So, for example, algorithm checking the size of elements (TD1.1, TD1.2), their colour (TD2.2, TD2.3), the distances (TD1.5), the automatic reorganization (TD1.6), the automatic highlighting (TD6, TD6.1), *etc.* aren't visible in the diagrams of the previous section. One can only see their results. However some recommendations like the navigational map, the current context, the legend, the names and types of the different diagrams of previous section are visible in the paper version.

However, a recommendation could still be illustrated with the *Conference Management System* example: the index (Recommendation TD16). But, in our case, it will be limited to a single type of diagram: the *Late requirements* one because the others aren't illustrated in the *Conference Management System* example. An example of such an index is visible in Table 9.3. Constructs are ordered by type and name. With this index one can easily find where the currently analysed construct can be found in the whole project.

Most recommendations for modellers are present in the example excepted recommendations MO7, MO8 and MO9.

The first one is related to the usage of comments. As one can notice, there is no comment in diagrams of Figures 9.4, 9.5 and 9.6. We decided to not introduce it because doing so would make our example artificial.

The recommendation MO8 was already discussed in the paragraph concerning language engineers. Indeed, recommendation MO8 advises to document models. Thus we should make a complete documentation of our model. However, it wasn't made because, in our example, we consider a single part, the *Late requirements* phase, of the complete *Tropos* project. Consequently, it was difficult to make a complete documentation of the *Conference Management System*.

The last recommendation was already discussed in Chapter 7. As it is a general recommendation, there is nothing special concerning our specific *Conference*

Table 9.3: *Conference Management System* index example

Name	Type	Diagrams names and types
Author	Actor	Actor model: Late req. CMS System (Goal Model): Late req.
CMS System	Actor	Actor model: Late req. CMS System (Goal Model): Late req. Coordinate Conference Analysis: Late req.
Program Committee (PC)	Actor	Actor model: Late req. CMS System (Goal Model): Late req. Coordinate Conference Analysis: Late req.
Publisher	Actor	Actor model: Late req. CMS System (Goal Model): Late req.
Reviewer	Actor	Actor model: Late req. CMS System (Goal Model): Late req.
Check availability	Hardgoal	Actor model: Late req.
Coordinate conference	Hardgoal	Actor model: Late req. CMS System(Goal Model): Late req. Coordinate Conference Analysis: Late req.
Deal with review	Hardgoal	Actor model: Late req. CMS System(Goal Model): Late req.
Get proceedings	Hardgoal	Actor model: Late req. CMS System(Goal Model): Late req.
Handle final paper	Hardgoal	CMS System(Goal Model): Late req.
Manage decision	Hardgoal	Coordinate Conference Analysis: Late req.
Manage proceedings	Hardgoal	Coordinate Conference Analysis: Late req.
Manage reviews	Hardgoal	Coordinate Conference Analysis: Late req.
Manage submissions	Hardgoal	Coordinate Conference Analysis: Late req.
Publication	Hardgoal	Actor model: Late req.
Publish proceedings	Hardgoal	Actor model: Late req.
Support paper submission	Hardgoal	Actor model: Late req. CMS System (Goal Model): Late req.
Notification	Resource	CMS System(Goal Model): Late req.
Get paper	Plan	Actor model: Late req.

Management System example.

9.3 Conclusion

In Sections 9.2.1 and 9.2.2, we respectively analyzed the current version of the *Tropos* diagrams of to the *Conference Management System* example and the modified versions of the same diagrams. Here, we show the benefits of the modification made in Section 9.2.2.

The weaknesses related to the discriminability principle of diagrams of Sec-

tion 9.2.1 was that some elements like PC and Author of Figure 9.2 were, according to [21], too small, others were too close from each others. Now, in the different diagrams of Section 9.2.2, those problems have been solved. In our case, those improvements aren't spectacular but, if we had reduced the size of Figure 9.2 one would require less attention to see the different actors and the borders among the different constructs with the new versions of diagrams than with the original versions.

For the modularity principle, the weakness of diagrams presented in [22] was that there were too much elements. This problem was not completely solved because it is difficult to divide the different diagrams in smaller parts. Indeed, we had to choose between dividing the diagrams according to the *"7 plus or minus 2"* rule quoted in [21] and a difficult cognitive integration. Having a lot of diagrams requires more concentration for the integration. Consequently, we decided to have diagrams with (a bit) more elements but a different structure. Indeed, *structuring* is "an *alternative* and a *complement* to decomposition" [19]. For example, in Figure 9.4, the *dependum*, eg. Check availability is situated between the *depender*, PC and the *dependee*, Reviewer. This way, one will understand that there is a link between the *dependum* and both surrounding actors.

The problem related to the emphasis principle presented in Table 9.1 is that the sizes of PC and Author actors of the diagram of Figure 9.2 are smaller. In the diagram of Figure 9.4, all constructs of the same type have the same size. This way, according to [21], one will consider the different elements as having the same importance. In the modified version, one won't have the impression that PC and Author actors are less important than others.

We also pointed out the fact that there is no highlighting for the CMS System actor in the actor model of Figure 9.2. We consider it as important because this diagram is part of the *Late requirements* phase and, in this phase, the most important element is the *system-to-be*. So, in the diagram of Figure 9.4, the CMS System actor is highlighted with a thicker border. This highlighting tip was presented in [21]. With this thicker border, one is attracted by the CMS System construct and will focus on it.

Concerning the principle related to the cognitive integration, two elements were added in diagrams of Section 9.2.2: a navigational map and navigation cues.

A navigational map improves the perceptual integration [21]. Such a map makes possible "to see the entire network of diagrams and navigational paths between them: it shows how the information space is broken up into viewable chunks" [21]. In Figure 9.7, one can see where one is in the complete network of diagrams with the highlighting of the current diagram (1.1 Actor model). One can also read that there are other diagrams under the current one with the links between 1.1 Actor model and 2.1 CMS Syst.(GM), 2.2 Reviewer (GM), *etc.* This way, one knows that there are other diagrams related to the one one is currently reading. This navigational map gives the complete network of diagrams and one hasn't to try to understand the different links oneself.

An element referring to conceptual integration was added: navigation cues. As it was explained in Section 9.2.2, those clues are present at two different places: with the navigational map and with the different *Tropos* steps. In the

part of the diagram dedicated to the different *Tropos* phases, those clues can be used to go backward and forward in the different phases. This way, one can directly navigate from a phase to the next or previous one, enabling one to see the logical links between diagrams of both phases. And if one doesn't remember the semantic of an element presented in the previous phase, one can easily go backward to reread it. It facilitates the transition between the different phases.

On an other hand, the navigation cues associated with the navigational map permits to navigate among the diagrams of a specific *Tropos* phase. This way, if one forgets some elements presented in a diagram of a higher level, a parallel one, *etc.*, one can easily go to this diagram and reread the required information. The goal of this feature is to facilitate user integration of information by improving navigation among diagrams.

Note that the index presented in Section 9.2.3, enables one to see the links among the different diagrams of the different phases. Indeed one can refer to it in order to see where a specific element is situated. This way, if one requires more information about a given element, one can directly see where one can find it. It enables the navigation among the different diagrams of the different *Tropos* phases in a less structured way than by using navigational map or *Tropos* phases cues.

Two elements were added according to the perceptual directness principle: icons and a tree layout. The icons should help in understanding the different actors. Indeed, in Section 9.2.2, we introduced two icons: a little man for *actors* and a computer for *system actors*. This should help to distinguish both types of actors. However, those icons were chosen arbitrarily and depend on the culture of the reader.

The second element introduced is a tree layout and is visible in Figure 9.6. This diagram has to be compared with the diagram of Figure 9.3. In this diagram, manage submissions, manage reviews, manage decision and manage proceedings are positioned at different Y positions. While in Figure 9.6, all those *Hardgoals* have the same vertical position. This way one knows that they are of the same level of decomposition. One doesn't have to read the diagram carefully to understand this property, it is visible in the tree layout. Consequently, one is less encline to miss this property.

The major modification about the structure principle is that PC and Author constructs of the diagrams of Section 9.2.2 have the same size as other actors. This way, those elements will be considered the same as the other ones. Their size is no more a discrimination variable according to [21].

Recommendation TD7 wasn't illustrated with our example. However, a square delimiting a region is more tangible than the use of proximity. Indeed, the interpretation of proximity for grouping elements hardly depends on personal perception. While elements situated in a "physically" closed area will be interpreted as belonging to a specific group.

In Chapter 7, we recommended to print diagram name and type when it is exported or printed. We applied this recommendation in diagrams of Section 9.2.2. Now, with the type of the diagram, one always knows where the diagram one is currently visualizing is situated in the global *Tropos* process. In our example, one knows that one is reading *Late requirements* diagrams and,

consequently, one knows that one has to focus on the *system-to-be*. However, one has to know/learn the role of the different *Tropos* phases. Moreover, one always knows the order of those phases with our representation which shows the order (and the names) of the different phases.

As it was explained in Section 9.2.2, two titles are visible on printed and exported diagrams: the name of the project and the name of the current diagram. The first name is useful when one is working on different projects at the same time. This information permits to directly know the name of the project. One would have to find the information oneself if it wasn't visible. It implies that one would loose time, require more attention, *etc.* The name of the current diagram is also important. Indeed, according to [21], it is the link with the represented world. This title gives a global overview of the diagram.

In diagrams of Section 9.2.2, a legend was also added. This legend is visible at the bottom left of each diagram (Part B of Figure 9.4). This artifact simplifies readers' work. Indeed, at the beginning, one probably doesn't know the graphical conventions of *Tropos*. This legend can help one. Then, one will, by experience, remember the graphical conventions. It implies that this legend should be optional, for example, when readers are used to *Tropos* notation. In diagrams of Section 9.2.2, legends are updated according to elements represented on the diagram. Consequently, one has the information one requires, and only it. One isn't disturbed by useless information and will avoid wasting time reading information one won't use.

The two last principles aren't really represented in our example. However, we can mention that the icons introduced for *Actor* and *System actor* constructs are new visual variables. Indeed, the icon is an extra variable that can be used in order to distinguish both types of actors. The visual expressiveness is thus, according to the visual expressiveness principle [21], increased.

9.4 Summary

The goal of this chapter was to show the benefits of the recommendations made in Chapters 6, 7 & 8. We started by introducing the *Conference Management System* example in Section 9.1. Then, in Section 9.2, we presented the diagrams as discussed in [22] and the ones modified according to our recommendations. We also explained why some recommendations can't be illustrated with the *Conference Management System* example. Finally, in Section 9.3, we compared both versions of diagrams presented in Sections 9.2.1 and 9.2.2 and showed elements which are, according to us, improved.

Chapter 10

Validating recommendations with tool developers

In the previous chapter, we validated our recommendations with the *Conference Management System* example. The goal was to show that recommendations proposed in Chapters 6, 7 & 8 really improve the effectiveness of diagrams.

In this chapter, we present another validation type: the validation of recommendations for tool developers made by *TAOM4e* developers. Those people know *TAOM4e* very well and are consequently able to give their opinion about the different recommendations made in Chapter 7. Those recommendations would directly impact the way modellers create models. The objective is to prioritise the recommendations in order to enable their selection for implementation.

In order to explain the entire process which allows us to get modellers opinion, we first present the interviewed people in Section 10.1. Then, in Section 10.2, we explain the objective of the evaluation by tool developers. In the next section, we present the evaluation form. Then, in Section 10.4, we check the validity of our evaluation. In Section 10.5, we give the results of the survey. And, finally, we analyse those results in Section 10.6.

Note that the identifiers used here for the recommendations are those presented in Section 7.19.

10.1 Respondents

We selected 4 different developers of *TAOM4e*. It implies that they participated in its development or, at least, to its maintenance and improvement. It means that they know the tool more deeply than any other users. They are aware of what is behind the tool, its weaknesses, *etc.* They often already know what could be improved according to the *Tropos* philosophy [6]. In consequence, they can also add some recommendations.

Moreover, those developers are also *TAOM4e* users. Consequently, they know this tool very well, strengths as well as weaknesses. They can thus easily say "that recommendation will be very useful", or, on the other hand, "it won't change modellers' experience". The extreme opinion is that a recommendation decreases the usability of the tool. In this case this extra functionality shouldn't be developed.

The interviewed group still has another advantage: all its members are working with different modelling languages and different tools. It implies that they aren't locked in a single language with a single tool. They use various tools frequently. It means that they have a lot of experiences with a range of software of the same family as *TAOM4e*. By consequent, they can easily give their opinion about the proposed new features which are already present in those other tools.

10.2 Research objective

The research objective is to find the most important recommendations according to tool developers. Those developers are generally also modellers.

Moreover our respondents are experienced with several modelling languages and tools, so they can recommend to not develop some functionalities they already used and consider as useless in other tools. Conversely, they are also able to propose other functionalities for *Tropos* tools, improve the proposed ones, *etc.*

10.3 Evaluation form

In order to get tool developers opinion, they were provided with an assessment form which is presented in Appendix A. There, one can see that there are 3 major columns: Recommendations, Value and Priority.

In the Recommendations column, the identifiers of the different recommendations for tool developers represent corresponding recommendations. Obviously, a list like the one presented in Section 7.19 was given to interviewed people in order to get their complete understanding of those identifiers.

The next column, Value, is divided into two sub-columns: Mandatory and Optional. When a recommendation is declared as Mandatory, it means that the person taking part in the poll thinks that it should really be implemented, while Optional recommendations could be added but it is not really necessary according to her/him.

The last column, Priority, is divided into three sub-columns: High, Medium and Low. It is necessary because it permits one to have a deeper classification than simply dividing proposals in Mandatory and Optional classes.

This form was thus given to the different respondents who filled it in individually. It implies that each evaluation was made independently from the others. Moreover, we weren't present when users gave their opinion. In this way, we didn't influence their choices.

10.4 Validity of the evaluation

As it is explained upper, this survey was carried out with four *TAOM4e* developers. This number can seem small and one can wonder if the results of this survey are generalisable.

The point is that the four respondents know *TAOM4e* very well as they took part in its development or, at least, its maintenance. If we had interviewed other people, their knowledge of the tool would have been less accurate. The evaluation form sometimes focuses on very specific aspects of *TAOM4e*, consequently, only advanced users could answer this survey. The individual results would probably be slightly different if we had chosen more users with less knowledge of *TAOM4e*. However, global results would probably converge on the same opinions as with our 4 respondents. We can say that we preferred quality over quantity. It allowed us to have the opinion of experts who have a complete knowledge about the tool.

10.5 Results of the evaluation

The synthesis of the results of the survey can be found in Table 10.1. The table must be read as follows: lines represent the different recommandations listed in Section 7.19 and the columns must be associated by pairs. Each pair represents the opinion of a single person. The name of the different people is obviously hidden. The two last columns (in bold) are an attempt of summary of the different opinions from the 8 previous columns.

10.6 Results analysis

In the previous section, we mentioned that the two last columns are an attempt of summary because some opinions are quite different and we had to make some choices.

A very special situation can be found at the line concerning recommendation TD1.5. It is related to the minimal distance between the different constructs. The three people who declared this advice as *mandatory* and *high* probably understood the problem of elements which are too close from each others. But it doesn't mean that the fourth person didn't understood it. S/he can have a lot of other reasons to declare it as optional and low. S/he can consider that such a mechanism will imply diagrams requiring huge surfaces, *etc.*

Lines TD2.2 and TD2.3 are quite similar. The opinions are divided in two equal parts: one thinks that those recommendations are mandatory while the

Table 10.1: Summary of the opinions

	Value A	Priority A	Value B	Priority B	Value C	Priority C	Value D	Priority D	**Synthesis value**	**Synthesis priority**
TD1	-	-	-	-	-	-	-	-	-	-
TD1.1	Mandatory	High	Mandatory	High	Mandatory	Low	Mandatory	High	**Mandatory**	**High**
TD1.2	Mandatory	High	Mandatory	High	Mandatory	Low	Mandatory	Low	**Mandatory**	**Medium**
TD1.3	Optional	Medium	Optional	Medium	Optional	Low	Optional	Low	**Optional**	**Medium**
TD1.4	Optional	High	Optional	High	Optional	Low	Optional	Low	**Optional**	**Medium**
TD1.5	Mandatory	High	Mandatory	High	Optional	Low	Mandatory	High	**Mandatory**	**High**
TD1.6	Mandatory	High	Mandatory	High	Mandatory	Medium	Mandatory	Medium	**Mandatory**	**High**
TD2	-	-	-	-	-	-	-	-	-	-
TD2.1	Mandatory	High	Mandatory	High	Mandatory	Medium	Optional	Medium	**Mandatory**	**Medium**
TD2.2	Mandatory	High	Mandatory	High	Optional	Low	Optional	Medium	**Mandatory**	**Low**
TD2.3	Mandatory	High	Mandatory	High	Optional	High	Optional	Medium	**Mandatory**	**Low**
TD3	Mandatory	High	Mandatory	High	Mandatory	High	Mandatory	Medium	**Mandatory**	**High**
TD4	Mandatory	High	Mandatory	High	Mandatory	Low	Mandatory	Medium	**Mandatory**	**Medium**
TD5	Optional	Medium	Optional	Medium	Mandatory	Medium	Optional	Low	**Optional**	**Medium**
TD6	Mandatory	High	Mandatory	High	Mandatory	Medium	Mandatory	High	**Mandatory**	**High**
TD6.1	Mandatory	High	Mandatory	High	Mandatory	Medium	Mandatory	High	**Mandatory**	**High**
TD7	Optional	Medium	Optional	Medium	Mandatory	High	Optional	High	**Optional**	**High**
TD8	Mandatory	High	Mandatory	High	Mandatory	Low	Mandatory	High	**Mandatory**	**High**
TD8.1	Optional	High	Optional	High	Mandatory	Low	Mandatory	Medium	**Optional**	**High**
TD8.2	Optional	High	Optional	Medium	Mandatory	Low	Mandatory	Medium	**Optional**	**High**
TD8.3	Mandatory	Medium	Mandatory	Medium	Mandatory	Low	Optional	Low	**Mandatory**	**Low**
TD9	Optional	High	Optional	High	Optional	Low	Mandatory	Medium	**Optional**	**High**
TD10	Optional	Low	Optional	Low	Optional	Medium	Optional	Medium	**Optional**	**Medium**
TD11	Optional	Medium	Optional	Medium	Optional	Medium	Optional	Low	**Optional**	**Medium**
TD12	Optional	Medium	Optional	Medium	Optional	High	Mandatory	Medium	**Optional**	**High**
TD13	Optional	High	Optional	High	Optional	Medium	Optional	Low	**Optional**	**Medium**
TD14	Optional	High	Mandatory	High	Mandatory	High	Mandatory	High	**Mandatory**	**Medium**
TD15	Mandatory	Medium	Mandatory	Medium	Mandatory	High	Optional	High	**Mandatory**	**Medium**
TD16	Mandatory	Medium	Mandatory	Medium	Mandatory	Medium	Optional	Medium	**Mandatory**	**Medium**
TD16.1	Optional	Medium	Optional	High	Optional	Low	Optional	Medium	**Optional**	**Medium**
TD17	Optional	Medium	Optional	Medium	Optional	Medium	Optional	Low	**Optional**	**Medium**

other thinks the opposite. What can we think about it? Probably, simply consider that the opinion depends on a large set of parameters like the frequency of usage of the tool, the experience with other tools, the personal view of the feature, *etc.*

There are still other examples like the ones presented here above but we don't analyse all of them. The fact is that some choices had to be made in case of conflict. It means that those synthetic values and priorities aren't always extrapolated from the different opinions. There is a part of subjectivity but we tried to minimize it as much as possible.

The result of this survey is that we can classify the different proposals in six categories corresponding to the six different values given by the different respondents. This classification is provided in Table 10.2.

Table 10.2: Summary of recommendations evaluation

Value \ Priority	Low	Medium	High
Optional	/	TD1.3 TD1.4 TD5 TD10 TD11 TD13 TD16.1 TD17	TD7 TD8.1 TD8.2 TD9 TD12
Mandatory	TD2.2 TD2.3 TD8.3	TD1.2 TD2.1 TD4 TD14 TD15 TD16	TD1.1 TD1.5 TD1.6 TD3 TD6 TD6.1 TD8

Obviously, the recommendations classified as Mandatory and High should be implemented in priority because interviewed people consider them of high importance. This prioritisation should thus also help for the implementation order.

10.7 Design and implementation

A logical continuation would be an implementation of the proposed recommendations for tool developers. Some features were designed and implemented but are not presented here. Information can be found in Appendix B.

10.8 Summary

In this Chapter, we presented the validation of recommendations for tool developers. This validation was made by some *TAOM4e* developers. We first presented those interviewed users. Then, we defined the objective of this prioritisation of the different recommendations for tool developers. We also presented the evaluation form. Finally, we presented and analysed the results of this survey. The results of this process can be used to determine an implementation order.

Part IV

Conclusion

Chapter 11

Conclusions and future work

In this last chapter, we first present the major conclusions of the work. Then, we present the limitations of the approach used. Finally, we explain what could be the continuation of the presented work.

11.1 Conclusions

The starting hypothesis of this thesis was that diagrams used in requirements engineering act as a barrier rather than an aid to user-developer communication [21]. However, the goal of software systems requirements engineering languages is to clearly understand the needs of the users in order to build software systems which match with their needs.

In [21], Moody presents a set of 9 principles for effective communication which aim at improving the communication between users and developers. These principles aren't drawn from scratch but relies on disciplines like cartography, conceptual modelling, cognitive psychology, communication theory, computer graphics, diagrammatic reasoning, education, graph drawing, human-computer interaction, information visualisation, linguistics, *etc.* This way, they rely on well-established theories of different disciplines which can influence diagrams effectiveness.

In our thesis, we applied these principles on a specific goal modelling language: *Tropos* and its supporting tool, *TAOM4e*. It means that we tried to find which elements presented in principles for visual effectiveness are present in this goal modelling language. But we also pointed out some limitations of this modelling approach. For example, we noticed that most default colours used in *TAOM4e* respect the different principles of cognitive effectiveness; that the *"holds" relationship* is an artifact for structuring information and reduce complexity of diagrams, *etc.* But we pointed out the fact that the size of the different constructs is determined by the length of their label, that the position of elements depends on modellers' preferences, that there is no help to divide diagrams in manageable chunks, that some elements related to principles are visible in *TAOM4e* but not in printed versions of diagrams, *etc.*

After this analysis, we noted that *Tropos/TAOM4e* could be improved *wrt* all principles. Consequently, we proposed recommendations which, according to us, improve the effectiveness of diagrams. The result of this analysis phase in our approach is a set of 3 recommendations lists.

The first list is meant for language engineers. Those people define the *Tropos* language and, consequently, can add new elements or improve existing ones in order to satisfy principles for cognitive effectiveness. So we recommended to write a clear manual for *Tropos*, to use different default colours for the actor constructs which use the same, to introduce icons, to provide mechanisms helping to divide diagrams in manageable chunks, to define navigation cues and to introduce techniques to group elements.

The second list is dedicated to people who develop tools which, like *TAOM4e*, enable one to draw *Tropos* diagrams. Check the unicity of the colour for the constructs of the same type, provide a mechanism for highlighting elements, provide a legend, make the information available in the tool visible on printed diagrams, provide a manual and/or a tutorial, *etc.* are examples of these recommendations.

Finally, the last list is a set of recommendations for modellers who use the language and the tool and who communicate directly with users of the *system-to-be*. We advised them to pay attention to the distance between elements, to pay attention to their size, to highlight most important elements, to document their models, to learn the tools and language principles from manuals, *etc.*

To validate our proposal we used two techniques. First, we illustrated recommendations with an example, the *Conference Management System*. There, we compared diagrams as they were drawn by *Tropos* modellers and their modified versions made according to our recommendations. In the second version of diagrams, we were able to see that the information is structured in such a way that the cognitive effectiveness of diagrams is improved. For example, we pointed out the fact that the navigational map helps to understand the decomposition of a diagram, that the legend helps novices and that the title and the type of a diagram help to understand its link with the modelled domain.

Second, a validation of the recommendations for tool developers was also carried out with *TAOM4e* developers. The conclusion of this validation is a classification of the different recommendations for tool developers in six categories. Seven recommendations should, according to them, be developed because they are prioritised with a high importance. These recommendations are: check that elements have a minimal size, check that there is a minimal distance between elements, provide a mechanism which automatically reorganizes elements, hide & show decomposition trees, provide a mechanism for highlighting most important elements, allow one to personalize this mechanism and provide a legend on each diagram. Another conclusion of this survey is that no recommendation is considered as useless.

We hope that the results of our approach will concretely improve the cognitive effectiveness of *Tropos* diagrams, particularly those drawn with *TAOM4e*.

11.2 Limitations

The work presented in this document still has some limitations which are explained here.

A first limitation is related to the analysis of *Tropos / TAOM4e wrt* principles for cognitive effectiveness. The point is that the analysis was carried out by a single person, the author of this thesis. Consequently, a part of subjectivity could be found in our understanding of the nine principles. Indeed, as we had no opportunity to confront our point of view, we had to rely on ourself.

Moreover, the analysis of *Tropos* can also be considered as objective as it has been made by a single person. Someone else would maybe analyse the current version of *Tropos / TAOM4e* in another way, depending on a set of parameters like the knowledge of *Tropos* and *TAOM4e*. Consequently, some language or tool properties were probably not observed.

One can also consider that our analysis is skewed by the example chosen for the analysis. Moreover one can also point out the fact that the analysis would be more accurate if the example was a real-size one.

Our validation of recommendations made by modellers also has some limitations.

Indeed, only 4 tool developers were questioned. One can consider that the evaluation team is too small. However, the respondents were specialists in the domain of evaluated elements, so we could gain the information and experience coming from the real users and developers of *TAOM4e*.

A second limitation of this validation is that it was made with our recommendations, not their implementation. Consequently, the respondents might have a different understanding of the proposed features. If recommendations for tool developers were implemented, the different opinions would be less subjectives.

11.3 Future work

Our future work includes the implementation of the different recommendations for tool developers presented in Chapter 7. This way, one would be able to see the changes generated by the different recommendations. In order to determine the implementation order, we could use the priorities of the different respondents of our survey of Chapter 10.

As the analysis of *Tropos* was made using a single tool, *TAOM4e*, we can also analyse this language with other tools. Consequently, we would have a different view of *Tropos* itself. In this case, we would be able to have a different opinion of the language. Moreover, the analysis of another tool would also result in recommendations specific to this tool.

We can still have a broader view in terms of languages. Indeed, in this thesis, we only presented the analysis of a single language: *Tropos*. Our future work would also include the analysis of other goal modelling languages like *i** and *NFR*.

Moreover, as it was explain in Section 11.2, the analysis and validation were made with small examples. So, in order to be more complete, we should also apply our analysis on a larger scale example.

Part V

References

Bibliography

[1] A. I. Antón. Goal-Based Requirements Analysis. In *IEEE International Conference on Requirements Engineering (ICRE 96)*, pages 136–144, 1996.

[2] Association for Computing Machinery. *ACM SIG Conference Manual*, Last accessed April 2008.

[3] J. Bertin. *Semiology of Graphics: Diagrams, Networks, Maps*. University of Wisconsin Press, 1983.

[4] D. Bertolini, A. Novikau, A. Susi, and A. Perini. TAOM4E: an Eclipse ready tool for Agent-Oriented Modeling. Issue on the development process. Technical report, Fondazione Bruno Kessler - IRST, 2005.

[5] A. Bonetti. *Si* User's Guide*, 2007.

[6] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[7] P. P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[8] L. Chung, B. A. Nixon, J. Mylopoulos, and E. Yu. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.

[9] DIA. URL: http://www.gnome.org/projects/dia/.

[10] M. S. Feather, S. Fickas, A. Finkelstein, and A. van Lamsweerde. Requirements and Specification Exemplars. *Automated Software Engineering*, 4:419–438.

[11] P. Giorgini, J. Mylopoulos, A. Perini, and S. Angelo. The Tropos Metamodel and its Use. *Informatica*, 29(4):401–408, 2005.

[12] Institute of Electrical and Electronics Engineers. *IEEE Conferences Organization Manual*, Last accessed April 2008.

[13] M. Jackson. *Problem Frames: Analysing and Structuring Software Development Problems*. Addison-Wesley, 2001.

[14] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[15] E. Kavakli and P. Loucopoulos. Goal Modeling in Requirements Engineering: Analysis and Critique of Current Methods. In *Information Modeling Methods and Methodologies*, pages 102–124. Idea Group, 2005.

[16] E. Letier. *Reasoning about Agents in Goal-Oriented Requirements Engineering*. PhD thesis, Université Catholique de Louvain (UCL), 2001.

[17] R. Matulevicius and P. Heymans. Visually Effective Goal Models Using KAOS. In *ER Workshops*, pages 265–275, 2007.

[18] G. Miller. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information.

[19] D. Moody. Dealing with "Map Shock": A Systematic Approach for Managing Complexity in Requirements Modelling. *REFSQ*, 2006.

[20] D. Moody. What Makes a Good Diagram? Communicating Effectively Using Diagrams. Staff-seminar presentation at the University of Namur, 2006.

[21] D. Moody. What Makes a Good Diagram? Improving the Cognitive Effectiveness of Diagrams in IS Development. *Advances in Information System Development*, 2006.

[22] M. Morandini, D. C. Nguyen, A. Perini, A. Siena, and A. Susi. Tool-supported Development with Tropos: The Conference Management System Case Study. In *Proceeding of 8th International Workshop on Agent Oriented Software Engineering*, 2007.

[23] B. Nuseibeh and S. Easterbrook. Requirements Engineering: A Roadmap. In *ICSE - Future of SE Track*, pages 35–46, 2000.

[24] H. S. Nwana. Software Agents: An Overview. *Knowledge Engineering Review*, 11(2):205–244, 1995.

[25] Object Management Group (OMG). *Unified Modeling Language: Superstructure version 2.0*, 2004.

[26] M. Petre. Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming. *Communications of the ACM*, 38(6):33–44, 1995.

[27] F. Sannicoló, F. Giunchiglia, and A. Perini. The Tropos Modeling Language. A User Guide. Technical report, Fondazione Bruno Kessler - IRST, 2002.

[28] G. Saval. Glossary of Conference Management, December 2006.

[29] The Standish Group. Chaos report. Technical report, The Standish Group, 1998.

[30] University of Toronto. *Organization Modeling Environment (OME): User Manual*.

[31] A. van Lamsweerde. The KAOS Meta-model: Ten Years After. Technical report, Université Catholique de Louvain (UCL), 2003.

[32] E. Yu. Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. In *3rd IEEE International Symposium on Requirements Engineering (RE'97)*, pages 226–235, 1997.

Part VI

Appendix

Appendix A

Evaluation form

Table A.1: Evaluation form

Recommendations	**Value**		**Priority**		
	Mandatory	**Optional**	**High**	**Medium**	**Low**
TD1	-	-	-	-	-
TD1.1					
TD1.2					
TD1.3					
TD1.4					
TD1.5					
TD1.6					
TD2	-	-	-	-	-
TD2.1					
TD2.2					
TD2.3					
TD3					
TD4					
TD5					
TD6					
TD6.1					
TD7					
TD8					
TD8.1					
TD8.2					
TD8.3					
TD9					
TD10					
TD11					
TD12					
TD13					
TD14					
TD15					
TD16					
TD16.1					
TD17					

Appendix B

Implemented features

We implemented functionalities which seemed reachable according to our knowledge of *GMF*. Those features can be divided in two groups: the layouts and the extension of the selection algorithms. The first group contains an algorithm which reorganizes the selected elements in a tree layout while the second contains two algorithms enabling the selection extension according to the nodes currently selected and some parameters.

B.1 Layouts

B.1.1 Tree layout

B.1.1.1 Presentation

This layout organizes selected nodes as a tree. Selected elements with no outgoing link (or outgoing links linked to elements which are not selected) are considered as the roots and positioned at the top. Selected elements having links outgoing only to root elements (and eventually to non selected elements) are placed at the second level. And so on.

Note that this algorithm has some weaknesses. One of them is that it doesn't care about the position of elements which aren't selected. It means that it can superimpose one (or more) selected element(s) over another which isn't selected. It also implies that there is no check for the minimal distance property, *etc.*

B.1.1.2 Basic principles

1. **Classify selected elements in different layers**
   The method *findLevels(Vector levels, Vector leaves)* aims at classifying selected elements incrementally in different levels (roots first, *etc.*). We iterate through the vector of selected nodes. For each node we check if connections are outgoing from it. If there is no outgoing link it means that

the node is a root[1]. Otherwise we look at the target of each connection outgoing from the analysed node. If all the targets are either a non selected node, either already classified in an upper level it means that the node can be added to the current level because all its "ancestors" have already been classified. If there is one link (or more) which doesn't fill none of those two conditions then the current node is placed in a vector of nodes which will be used for a further call to the function.
When we have iterated through all elements, we check if at least one node has been added to the current level. If the answer is negative it means that there is a cycle. It is consequently impossible to determine levels. Otherwise, the current level is added into the levels vector after the previous level and we re-call the fonction with the vector of nodes which aren't still classified.
At the end, the function returns a vector containing the width used by each level or a vector containing a single element (-1) if there is a cycle.

2. **Determine the maximum width used by the layers**[2]
   The idea is to determine the maximum width used by the different layers. In order to do this, we iterate through the result vector of the *findLevels* method and take the largest width.

3. **Determine the starting X & Y positions**
   The idea is to determine the top-left point used by root elements. Then the positions of other elements will be computed from this point.

4. **Place the nodes on the diagram**
   Nodes are placed on the diagram according to the height and width of their level, the height of the upper levels, the width of previous elements of the same level, *etc.* This part of the code is enclosed in a special class in which enables the *Undo* method.

B.1.1.3 Installation steps

1. Add a menu element in the plugin.xml file

```
<extension point="org.eclipse.ui.actionSets">
    <actionSet
          id="taom4e.diagram.actionSet"
          label="Layout"
          visible="true">
       <menu
             id="layoutMenu"
             label="Layout">
          <separator
                name="layoutAlgorithm">
          </separator>
       </menu>
       <action
```

[1] This check will succeed only during the first call to the method which aims at finding roots

[2] The next 3 steps are activated iff there is no cycle in the levels.

```
            class="TroposCompact.diagram.actions.TreeLayout"
            icon="icons/sample.gif"
            id="taom4e.diagram.actions.TreeLayout"
            label="&amp;Tree layout"
            menubarPath="layoutMenu/layoutAlgorithm"
            toolbarPath="layoutAlgorithm"
            tooltip="Organizes elements according to a tree layout">
      </action>
   </actionSet>
</extension>
```

2. Insert the source code of the layout in the package referenced by the "plugin.xml" file

```
package TroposCompact.diagram.actions;

    import java.awt.Point;
    import java.util.Iterator;
    import java.util.LinkedList;
    import java.util.List;
    import java.util.Vector;
    import org.eclipse.core.commands.operations.OperationHistoryFactory;
    import org.eclipse.core.runtime.IAdaptable;
    import org.eclipse.core.runtime.IProgressMonitor;
    import org.eclipse.core.runtime.NullProgressMonitor;
    import org.eclipse.draw2d.IFigure;
    import org.eclipse.emf.ecore.EObject;
    import org.eclipse.emf.transaction.TransactionalEditingDomain;
    import org.eclipse.emf.transaction.util.TransactionUtil;
    import org.eclipse.emf.workspace.util.WorkspaceSynchronizer;
    import org.eclipse.gmf.runtime.common.core.command.CommandResult;
    import org.eclipse.gmf.runtime.diagram.ui.editparts.ConnectionEditPart;
    import org.eclipse.gmf.runtime.diagram.ui.editparts.IGraphicalEditPart;
    import org.eclipse.gmf.runtime.emf.commands.core.command.AbstractTransactionalCommand;
    import org.eclipse.gmf.runtime.notation.Bounds;
    import org.eclipse.gmf.runtime.notation.Node;
    import org.eclipse.jface.action.IAction;
    import org.eclipse.jface.dialogs.MessageDialog;
    import org.eclipse.jface.viewers.ISelection;
    import org.eclipse.jface.viewers.IStructuredSelection;
    import org.eclipse.ui.IWorkbenchWindow;
    import org.eclipse.ui.IWorkbenchWindowActionDelegate;
    import TroposCompact.diagram.edit.parts.HardGoalEditPart;
    import TroposCompact.diagram.edit.parts.PlanEditPart;
    import TroposCompact.diagram.edit.parts.ResourceEditPart;
```

```
import TroposCompact.diagram.edit.parts.SoftGoalEditPart;
import TroposCompact.diagram.part.TroposCompactDiagramEditorPlugin;


public class TreeLayout implements IWorkbenchWindowActionDelegate {
      private IWorkbenchWindow window;
      private Vector mySelection;
      private static final int XOFFSET=20;//Horizontal distance between tree elements
      private static final int YOFFSET=20;//Vertical distance between tree elements

      /**
       * The constructor.
       */
      public TreeLayout() {
      }

      private Vector findLevels(Vector levels, Vector nodes){//returns a Vector containing the widths of the current
level and lower ones
            Iterator current = nodes.iterator();
            Vector templevel=new Vector();
            Vector templeaves=new Vector();
            int width=0;
            while (current.hasNext()){
                  IGraphicalEditPart leaf = (IGraphicalEditPart) current.next();
                  boolean found=true;
                  if (leaf.getSourceConnections().isEmpty()){
                        found=true;
                  }
                  else{
                        Iterator ltargets = leaf.getSourceConnections().iterator();
                        while (ltargets.hasNext() && found){
```

```
found=false;
boolean currentfound=false;
IGraphicalEditPart target = (IGraphicalEditPart)((ConnectionEditPart) ltarg
    ets.next()).getTarget();

Iterator selected = mySelection.iterator();
boolean in=false;
while (selected.hasNext() && !in){
    IGraphicalEditPart temp=(IGraphicalEditPart) selected.next();
    if (!temp.equals(leaf) && temp.equals(target)){
        in=true;
    }
}
if (!in) currentfound=true;
else {
    Iterator currlayer=levels.iterator();
    while (currlayer.hasNext() && !currentfound){
        Iterator layer =((Vector)currlayer.next()).iterator();
        while (layer.hasNext() && !currentfound){
            IGraphicalEditPart elt=(IGraphicalEditPart) laye
    r.next();
            if (target.equals(elt)) currentfound=true;
        }
    }
    found=currentfound;
}
}
if (found) {
templevel.add(leaf);
width+=leaf.getFigure().getBounds().getSize().width+XOFFSET;
}
```

```
else templeaves.add(leaf);
if(!current.hasNext()){
      if (templevel.isEmpty()) {
            MessageDialog.openInformation(
                  window.getShell(),
                  "Error",
                  "It is impossible to define layers because there is a cycle");
            Vector result=new Vector();
            result.add(-1);
            return result;
      }
      else {
            levels.add(templevel);
            if (templeaves.iterator().hasNext()) {
                  Vector result=new Vector();
                  result.add(width-XOFFSET);
                  Iterator i =findLevels(levels, templeaves).iterator();
                  while (i.hasNext()) {
                        Integer rslt=(Integer)i.next();
                        if (rslt==-1) {
                              result=new Vector();
                              result.add(-1);
                              return result;
                        }
                        else {
                              result.add(rslt);
                        }
                  }
                  return result;
            }
            else {
```

```
                Vector result=new Vector();
                result.add(width-XOFFSET);
                return result;
            }
        }
    }
    Vector result=new Vector();
    result.add(0);
    return result;
}

private Point gettopcorner(Vector lnodes){//returns the min x & min y positions of the roots
        int x=Integer.MAX_VALUE;
        int y=Integer.MAX_VALUE;
        Iterator i=lnodes.iterator();
        while (i.hasNext()){
            IGraphicalEditPart node=(IGraphicalEditPart)i.next();
            IFigure figure=node.getFigure();
            if (node.getFigure().getBounds().x<x) x=node.getFigure().getBounds().x;
            if (node.getFigure().getBounds().y<y) y=node.getFigure().getBounds().y;
        }
        if (x<0) x=0;
        return new Point(x,y);
}

/**
 * The action has been activated. The argument of the
 * method represents the 'real' action sitting
 * in the workbench UI.
 * @see IWorkbenchWindowActionDelegate#run
```

```
 */
    public void run(IAction action) {
            Vector levels=new Vector();
            Vector dimensions = new Vector();
            int gridTemp = 0;
            Vector widths=findLevels(levels, mySelection);
            if ((Integer)widths.elementAt(0)!=-1){
                    int rootwidth=(Integer)widths.elementAt(0);
                    Iterator i =widths.iterator();
                    while (i.hasNext()) {
                            int width=(Integer)i.next();
                            dimensions.add(width);
                            if(width>gridTemp) gridTemp=width;
                    }
                    EObject elt=(((IGraphicalEditPart)(((Vector)(levels.elementAt(0))).elementAt(0))).resolveSema
nticElement());
                    final int gridWidth=gridTemp;
                    final int startingX=((int)((gettopcorner((Vector)levels.elementAt(0))).getX()))-((gridWidth-r
ootwidth)/2);
                    final int startingY=(int)((gettopcorner((Vector)levels.elementAt(0))).getY());
                    final Iterator itlayers = levels.iterator();
                    final Iterator itdimensions=dimensions.iterator();
                    TransactionalEditingDomain editingDomain = TransactionUtil.getEditingDomain(elt);
                    List affectedFiles = new LinkedList();
                    affectedFiles.add(WorkspaceSynchronizer.getFile(elt.eResource()));
                    AbstractTransactionalCommand command = new AbstractTransactionalCommand(editingDomain, "Combi
ne Domains", affectedFiles)
                                    {
                                            @Override
                                            protected CommandResult doExecuteWithResult(IProgressMonitor monitor, IAdaptable inf
o)
```

```
{
    int y=startingY;
    while (itlayers.hasNext()){
        Iterator itcurrlayer=((Vector)itlayers.next()).iterator();
        int levelwidth=(Integer)itdimensions.next();
        int levelheight=0;
        int x=startingX +((gridWidth-levelwidth)/2);
        while (itcurrlayer.hasNext()){
            IGraphicalEditPart node=(IGraphicalEditPart)itcurrlayer.n
ext();

            Bounds bounds = (Bounds)((Node)node.getModel()).getLayout
Constraint();

            bounds.setX(x);
            bounds.setY(y);
            ((Node)node.getModel()).setLayoutConstraint(bounds);
            x+=node.getFigure().getBounds().width+XOFFSET;
            if (node.getFigure().getBounds().height>levelheight) leve
lheight=node.getFigure().getBounds().height;
        }
        y+=levelheight+YOFFSET;
    }
    return CommandResult.newOKCommandResult();
}
};
try
{
    OperationHistoryFactory.getOperationHistory().execute(command,new NullProgressMonito
r(), null);
}
catch (org.eclipse.core.commands.ExecutionException e)
```

```
        {
            MessageDialog.openError(window.getShell(), "error", e
                    .getMessage());
            TroposCompactDiagramEditorPlugin.getInstance().logError(
                    "error", e);
        }
    }
}
/**
 * Selection in the workbench has been changed. We
 * can change the state of the 'real' action here
 * if we want, but this can only happen after
 * the delegate has been created.
 * @see IWorkbenchWindowActionDelegate#selectionChanged
 */
public void selectionChanged(IAction action, ISelection selection) {
    mySelection = null;
    action.setEnabled(false);
    if (selection instanceof IStructuredSelection == false
            || selection.isEmpty()) {
        return;
    }
    mySelection=new Vector();
    Iterator it=((IStructuredSelection)selection).iterator();
    while (it.hasNext()){
        Object temp=it.next();
        if (temp instanceof IGraphicalEditPart){
            IGraphicalEditPart editPart = (IGraphicalEditPart) temp;
            if ((editPart instanceof PlanEditPart)|| (editPart instanceof HardGoalEditPart) || (
editPart instanceof SoftGoalEditPart) || (editPart instanceof ResourceEditPart))
```

```
                mySelection.add(editPart);
            }
        }
        if (!(mySelection.isEmpty())) action.setEnabled(true);
    }

    /**
     * We can use this method to dispose of any system
     * resources we previously allocated.
     * @see IWorkbenchWindowActionDelegate#dispose
     */
    public void dispose() {
    }

    /**
     * We will cache window object in order to
     * be able to provide parent shell for the message dialog.
     * @see IWorkbenchWindowActionDelegate#init
     */
    public void init(IWorkbenchWindow window) {
        this.window = window;
    }

}
```

B.2 Selection extension

B.2.1 Select all subelements

B.2.1.1 Presentation

This function is activated only when (at least) one node (*Plan*, *Hardgoal*, *Softgoal* or *Resource*) is selected and aims at selecting all the subelements (without distance limitation) of the selected node(s). A subelement is an element linked to the analysed element with a connection originating from itself.

B.2.1.2 Basic principles

The idea is to iterate through selected elements. If the selected element is a *Plan*, *Hardgoal*, *Softgoal* or *Resource* then we try to find its subelements. The subelements are elements which are linked to the analysed element by a link originating from this element and ending at the analysed element. Then we iteratively call this fonction using the subelements as the root elements. The analysis is finished when there is no more node to add.

B.2.1.3 Installation steps

1. Add a menu element in the plugin.xml file

```
<extension point="org.eclipse.ui.actionSets">
    <actionSet
          id="taom4e.diagram.selectSet"
          label="Select"
          visible="true">
       <menu
             id="selectMenu"
             label="Selection">
          <separator
                name="selectAlgorithm">
          </separator>
       </menu>
       <action
             class="TroposCompact.diagram.actions.SubelementsSelection"
             icon="icons/sample.gif"
             id="taom4e.diagram.actions.SubelementsSelection"
             label="&amp;Select all subelements"
             menubarPath="selectMenu/selectGroup"
             toolbarPath="selectGroup"
             tooltip="Select all subelements of selected node(s)">
       </action>
    </actionSet>
 </extension>
```

2. Insert the source code of the layout in the package referenced by the "plugin.xml" file

```
package TroposCompact.diagram.actions;

    import java.util.*;
    import org.eclipse.gmf.runtime.diagram.ui.editparts.ConnectionEditPart;
    import org.eclipse.gmf.runtime.diagram.ui.editparts.IGraphicalEditPart;
    import org.eclipse.jface.action.IAction;
    import org.eclipse.jface.viewers.ISelection;
    import org.eclipse.jface.viewers.IStructuredSelection;
    import org.eclipse.ui.IWorkbenchWindow;
    import org.eclipse.ui.IWorkbenchWindowActionDelegate;
    import TroposCompact.diagram.edit.parts.HardGoalEditPart;
    import TroposCompact.diagram.edit.parts.PlanEditPart;
    import TroposCompact.diagram.edit.parts.ResourceEditPart;
    import TroposCompact.diagram.edit.parts.SoftGoalEditPart;

    public class SubelementsSelection implements IWorkbenchWindowActionDelegate{
        private IWorkbenchWindow window;
        private Vector mySelection;

        public SubelementsSelection(){

        }

        private void explore(IGraphicalEditPart node){
            Iterator lsons = node.getTargetConnections().iterator();
            while (lsons.hasNext()){
                IGraphicalEditPart son=(IGraphicalEditPart)(((ConnectionEditPart)lsons.next()).getSource());
                if (son.getSelected()==0){
                    son.getViewer().appendSelection(son);
                    explore(son);
```

```
        }
    }
}

public void run(IAction action) {
    Iterator selection = mySelection.iterator();
    while (selection.hasNext()){
        IGraphicalEditPart editPart = (IGraphicalEditPart) selection.next();
        explore(editPart);
    }
}

public void selectionChanged(IAction action, ISelection selection) {
    mySelection = null;
    action.setEnabled(false);
    if (selection instanceof IStructuredSelection == false
            || selection.isEmpty()) {
        return;
    }
    mySelection=new Vector();
    Iterator it=((IStructuredSelection)selection).iterator();
    while (it.hasNext()){
        Object temp=it.next();
        if (temp instanceof IGraphicalEditPart){
            IGraphicalEditPart editPart = (IGraphicalEditPart) temp;
            if ((editPart instanceof PlanEditPart)|| (editPart instanceof HardGoalEditPart) || (
editPart instanceof SoftGoalEditPart) || (editPart instanceof ResourceEditPart))
                mySelection.add(editPart);
        }
    }
    if (!(mySelection.isEmpty())) action.setEnabled(true);
```

```
}

public void dispose() {

}

public void init(IWorkbenchWindow window) {
        this.window = window;
    }

}
```

B.2.2 Select linked elements

B.2.2.1 Presentation

This function selects all the elements surrounding the selected node(s) according to the maximum "distance" defined by the user. It is activated only if at least one node is selected. It is also possible to use an unlimited "distance". By "surrounding element", we mean element linked to the selected element(s) through a (chain of) connection link(s).

B.2.2.2 Basic principles

This function is similar to the method which aims at selecting all subelements excepted that the "distance" can be limited and that we also select parent elements (and not only subelements).

B.2.2.3 Installation steps

1. Add a menu element in the plugin.xml file

```
<extension point="org.eclipse.ui.actionSets">
    <actionSet
          id="taom4e.diagram.selectSet"
          label="Select"
          visible="true">
       <menu
             id="selectMenu"
             label="Selection">
          <separator
                name="selectAlgorithm">
          </separator>
       </menu>
       <action
             class="TroposCompact.diagram.actions.SubelementsSelection"
             icon="icons/sample.gif"
             id="taom4e.diagram.actions.SubelementsSelection"
             label="&amp;Select all subelements"
             menubarPath="selectMenu/selectGroup"
             toolbarPath="selectGroup"
             tooltip="Select all subelements of selected node(s)">
       </action>
       <action
             class="TroposCompact.diagram.actions.SelectAll"
             icon="icons/sample.gif"
             id="taom4e.diagram.actions.SelectAll"
             label="&amp;Select linked elements"
             menubarPath="selectMenu/selectGroup"
             toolbarPath="selectGroup"
             tooltip="Select elements linked to the selected node(s)">
       </action>
    </actionSet>
 </extension>
```

2. Insert the source code of the layout in the package referenced by the "plugin.xml" file
   (a) Plug-in code

```
package TroposCompact.diagram.actions;

import java.util.*;
import org.eclipse.gmf.runtime.diagram.ui.editparts.ConnectionEditPart;
import org.eclipse.gmf.runtime.diagram.ui.editparts.IGraphicalEditPart;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import TroposCompact.diagram.edit.parts.HardGoalEditPart;
import TroposCompact.diagram.edit.parts.PlanEditPart;
import TroposCompact.diagram.edit.parts.ResourceEditPart;
import TroposCompact.diagram.edit.parts.SoftGoalEditPart;

public class SelectAll implements IWorkbenchWindowActionDelegate{
	private IWorkbenchWindow window;
	private int delta;
	private Vector mySelection;

	public SelectAll(){

	}

	private void explore(IGraphicalEditPart node, int cpt){
		cpt+=1;
		Iterator lsons = node.getTargetConnections().iterator();
		while (lsons.hasNext()){
			IGraphicalEditPart son=(IGraphicalEditPart)(((ConnectionEditPart)lsons.next()).getSource());
			if (son.getSelected()==0){
				son.getViewer().appendSelection(son);
```

```
            if (cpt<delta) explore(son, cpt);
        }
    }
    Iterator lparents = node.getSourceConnections().iterator();
    while (lparents.hasNext()){
        IGraphicalEditPart parent=(IGraphicalEditPart)(((ConnectionEditPart)lparents.next()).getTarge
t());
        if (parent.getSelected()==0){
            parent.getViewer().appendSelection(parent);
            if (cpt<delta) explore(parent, cpt);
        }
    }
}

public void run(IAction action) {

    DistanceDialog dialog=new DistanceDialog(window.getShell());
    Double temp= (dialog.open());
    if (!(temp==null)){
        delta= temp.intValue();
        if (delta==-1) delta=Integer.MAX_VALUE;
        Iterator selection = mySelection.iterator();
        while (selection.hasNext()){
            IGraphicalEditPart editPart = (IGraphicalEditPart) selection.next();
            if (delta>0)explore(editPart,0);
        }
    }
}

public void selectionChanged(IAction action, ISelection selection) {
    mySelection = null;
```

```
        action.setEnabled(false);
        if (selection instanceof IStructuredSelection == false
                || selection.isEmpty()) {
            return;
        }
        mySelection=new Vector();
        Iterator it=((IStructuredSelection)selection).iterator();
        while (it.hasNext()){
            Object temp=it.next();
            if (temp instanceof IGraphicalEditPart){
                IGraphicalEditPart editPart = (IGraphicalEditPart) temp;
                if ((editPart instanceof PlanEditPart)|| (editPart instanceof HardGoalEditPart) || (
editPart instanceof SoftGoalEditPart) || (editPart instanceof ResourceEditPart))
                    mySelection.add(editPart);
            }
        }
        if (!(mySelection.isEmpty())) action.setEnabled(true);
    }

    public void dispose() {

    }

    public void init(IWorkbenchWindow window) {
        this.window = window;
    }

}
```

(b) Dialogue box

```
package TroposCompact.diagram.actions;

    import org.eclipse.swt.SWT;
    import org.eclipse.swt.graphics.Rectangle;
    import org.eclipse.swt.layout.GridData;
    import org.eclipse.swt.layout.GridLayout;
    import org.eclipse.swt.widgets.Button;
    import org.eclipse.swt.widgets.Dialog;
    import org.eclipse.swt.widgets.Display;
    import org.eclipse.swt.widgets.Event;
    import org.eclipse.swt.widgets.Label;
    import org.eclipse.swt.widgets.Listener;
    import org.eclipse.swt.widgets.Shell;
    import org.eclipse.swt.widgets.Text;

    public class DistanceDialog extends Dialog {
        Double value;

        /**
         * @param parent
         */
        public DistanceDialog(Shell parent) {
            super(parent);
        }

        /**
         * @param parent
         * @param style
         */
        public DistanceDialog(Shell parent, int style) {
            super(parent, style);
```

```
}

/**
 * Makes the dialog visible.
 *
 * @return
 */
public Double open() {
        Shell parent = getParent();
        final Shell shell =new Shell(parent, SWT.TITLE | SWT.BORDER | SWT.APPLICATION_MODAL);
        Rectangle bounds = shell.getBounds();
        int x = bounds.x + ((bounds.width - 110) / 2);
        int y = bounds.y + ((bounds.height - 90) / 2);
        shell.setLocation(x, y);

        shell.setText("Maximum distance");
        shell.setLayout(new GridLayout(2, false));

        Label label = new Label(shell, SWT.NULL);
        label.setText("Please enter distance:");

        final Text text = new Text(shell, SWT.SINGLE | SWT.BORDER);

        final Button checkBox = new Button(shell, SWT.CHECK);
        checkBox.setText("Not limited");
        GridData gridData = new GridData(GridData.HORIZONTAL_ALIGN_FILL);
        gridData.horizontalSpan = 2;
        checkBox.setLayoutData(gridData);

        final Button buttonOK = new Button(shell, SWT.PUSH);
        buttonOK.setText("Confirm");
```

```
buttonOK.setLayoutData(new GridData(GridData.HORIZONTAL_ALIGN_CENTER));
Button buttonCancel= new Button(shell, SWT.PUSH);
buttonCancel.setText("Cancel");
buttonCancel.setLayoutData(new GridData(GridData.HORIZONTAL_ALIGN_CENTER));

checkBox.addSelectionListener(new org.eclipse.swt.events.SelectionAdapter() {
    public void widgetSelected(org.eclipse.swt.events.SelectionEvent e) {
        boolean selected = checkBox.getSelection();
        if (selected == true) {
            text.setEnabled(false);
            value=new Double(-1);
            buttonOK.setEnabled(true);
        }
        else
        {
            text.setEnabled(true);
            try {
                value = new Double(text.getText());
                if (value>=0) buttonOK.setEnabled(true);
                else buttonOK.setEnabled(false);
            } catch (Exception excep) {
                buttonOK.setEnabled(false);
            }
        }
    }
});

text.addListener(SWT.Modify, new Listener() {
    public void handleEvent(Event event) {
        if (!checkBox.getSelection()){
```

```
            try {
                value = new Double(text.getText());
                if (value>=0) buttonOK.setEnabled(true);
                else buttonOK.setEnabled(false);
            } catch (Exception e) {
                buttonOK.setEnabled(false);
            }
        }
    });

    buttonOK.addListener(SWT.Selection, new Listener() {
        public void handleEvent(Event event) {
            shell.dispose();
        }
    });

    buttonCancel.addListener(SWT.Selection, new Listener() {
        public void handleEvent(Event event) {
            value = null;
            shell.dispose();
        }
    });

    shell.addListener(SWT.Traverse, new Listener() {
        public void handleEvent(Event event) {
            if(event.detail == SWT.TRAVERSE_ESCAPE)
                event.doit = false;
        }
    });
    text.setText("");
```

```
shell.pack();
shell.open();

Display display = parent.getDisplay();
while (!shell.isDisposed()) {
        if (!display.readAndDispatch())
                display.sleep();
}

return value;

    }

}
```